UNIVERSIDADE FEDERAL DE ITAJUBÁ

PROGRAMA DE PÓS-GRADUAÇÃO
EM CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Uma análise sobre as métricas para escalonamento dinâmico de processos em simulação distribuída usando protocolos otimistas

Emerson Assis de Carvalho

Itajubá, Agosto de 2013

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO
EM CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Emerson Assis de Carvalho

Uma análise sobre as métricas para escalonamento dinâmico de processos em simulação distribuída usando protocolos otimistas

Dissertação submetida ao programa de Pós-Graduação em Ciência e Tecnologia da Computação como parte dos requisitos para obtenção do título de Mestre em Ciência e Tecnologia da Computação.

Área de Concentração: Sistemas de Computação

Orientador: Prof. Dr. Edmilson Marmo Moreira
Co-orientador: Prof. Dr. Otávio A. S. Carpinteiro

Agosto de 2013
Itajubá

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS-GRADUAÇÃO
EM CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Emerson Assis de Carvalho

Uma análise sobre as métricas para escalonamento dinâmico de processos em simulação distribuída usando protocolos otimistas

Dissertação aprovada por banca examinadora em 22 de Agosto de 2013, conferindo ao autor o título de **Mestre em Ciência e Tecnologia da Computação**.

Banca Examinadora:
Prof. Dr. Edmilson Marmo Moreira (Orientador)
Prof. Dr. Otávio A. S. Carpinteiro (Co-orientador)
Prof^a. Dr^a. Sarita Mazzini Bruschi (USP-São Carlos)
Prof. Dr. Guilherme Souza Bastos

Itajubá
2013

Resumo

Esta dissertação apresenta uma avaliação sobre as métricas utilizadas pelos principais algoritmos de balanceamento de carga para simulação distribuída otimista que operam sobre o protocolo *Time Warp*. Propõe uma nova métrica e um algoritmo de balanceamento que opera independente da métrica utilizada, o que permite uma avaliação das métricas em um mesmo ambiente de execução, com os mesmos modelos de simulação e com os mesmos fatores de desbalanceamento de carga. A métrica proposta é baseada nos principais fatores que causam o desbalanceamento de uma simulação distribuída e o algoritmo foi projetado de maneira abstrata, visando não beneficiar qualquer uma das métricas. Considerando todos os modelos avaliados, a métrica proposta neste trabalho teve um bom desempenho, mas a métrica mais eficiente foi a métrica baseada na taxa de avanço do relógio local (LVT).

Abstract

This work presents a comparison of the metrics used by the main load balancing algorithms for optimistic distributed simulation executed under the Time Warp protocol. Introduces a new metric and a load balancing algorithm that is metric independent, allowing a metric evaluation under the same execution environment, with the same simulation models and the same load imbalance factors. The new metric proposed is based on the main load imbalance factors, and the load balancing algorithm was designed to not benefiting anyone of the metrics. With the models evaluated, the proposal metric archieved a good performance, but the most efficient metric was that based on the variance between processes Local Virtual Time (LVT).

Agradecimentos

Primeiramente, agradeço a Deus, por ter tantas pessoas especiais para agradecer, pela saúde e pela fé.

Aos meus pais, Noé Lourenço de Carvalho e Rosânia Fátima de Carvalho que, mesmo não entendendo grande parte do que eu faço, sempre confiaram em mim e apoiaram minhas escolhas.

À toda minha família, por apoiar e compreender minha ausência em momentos especiais.

Aos meus amigos, Anderson Oliveira Faria, Vanderlúcio de Araújo, Lênio Oliveira Prado Junior, Ricardo Emerson Júlio, Júlio César Almeida, Flávio Emílio Borges e Roberto Ribeiro Rocha, simplesmente pelas amizades verdadeiras.

Ao professor, Dr. Edmilson Marmo Moreira, pelos ensinamentos na graduação, por ter me aceitado, quando coordenador deste programa, como aluno de mestrado, pelas discussões e ensinamentos nas aulas da pós e, principalmente, pela confiança e empenho em me orientar neste trabalho.

Ao professor, Dr. Otávio Augusto Salgado Carpinteiro, por me co-orientar neste trabalho, pelos ensinamentos nas aulas da pós e pelo apoio financeiro conseguido junto à CAPES quando coordenador deste programa.

A todos os meus colegas de mestrado, em especial ao meu amigo Lênio Oliveira Prado Junior, pelo companheirismo constante nesses quase quatro anos de mestrado. Um agradecimento especial também ao amigo Roberto Ribeiro Rocha, pelas dicas de *Linux* e de programação.

À UNIFEI e a todos os seus funcionários, em especial aos demais professores da pós, pelos ensinamentos e sugestões em relação a este trabalho.

Ao LEMAF, especialmente ao coordenador de T.I, o Sr. Samuel Campos, por ter me liberado para as aulas da pós. Agradeço também a todos os meus antigos colegas de trabalho, que nunca questionaram minhas ausências.

À Devex, especialmente ao meu ex-diretor, o Sr. Luiz Thomaz do Nascimento, por ter me liberado para as aulas da pós. Agradeço também a todos os integrantes da minha equipe de trabalho mais recente, por terem suprido minha ausência com louvor. Agradeço em especial ao meu amigo Anderson Oliveira Faria, por ter realizado muitas das minhas tarefas gerencias enquanto me ausentava para realizar este trabalho.

A todos os meus professores da UNIFENAS, hoje meus atuais colegas de trabalho, pelos ensinamentos de outrora e pelo apoio no final deste trabalho. Um agradecimento especial aos professores Marcos Alberto de Carvalho e Alexandre Martins Dias.

Por fim, agradeço à CAPES, pelo apoio financeiro fornecido em um período deste trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

BGE Background Execution

CAT Cluster Advance Time

CPC Cluster-to-Processor Communication

CPU Unidade Central de Processamento

EEW Expected Effective Work

GVT Global Virtual Time

LVT Local Virtual Time

MPI Message Passing Interface

MRIP Multiple Replication in Parallel

NOW Network of Workstations

PAT Processor Advance Time

PCS Processor Communication Speed

PPC Processor-to-Processor Communication

SCC Strongly Connected Components

SD Sistema Distribuído

SGs Strong Groups

SRIP Single Replication in Parallel

STF Smallest Timestamp First

1 Introdução

O processo de desenvolvimento de sistemas complexos necessita de ferramentas de apoio ao desenvolvimento e de avaliação, pois é importante identificar a abordagem mais apropriada. As ferramentas de apoio são as soluções mais indicadas em situações onde a criação de protótipos pode ser difícil ou não apropriada em relação a tempo e custo (FUJIMOTO, 2000).

Uma simulação é um processamento que modela o comportamento de algum sistema e vem sendo cada vez mais utilizada devido a sua flexibilidade e baixo custo (AGARWAL; HYBINETTE; XIONG, 2005). Uma simulação resulta da modelagem de um processo ou sistema e permite realizar experimentos com um modelo, permitindo uma análise sobre seu comportamento.

A simulação vem sendo usada em diversas áreas, tais como: aplicações militares (simuladores de guerra, tanques, vôos), entretenimento (criação de mundos virtuais para jogos), educação (como o treinamento de médicos cirurgiões), na área de redes (avaliação de *hardwares*, *softwares* e/ou protocolos), desenvolvimento de circuitos digitais e sistemas computacionais, transportes (no controle de tráfego, por exemplo), manufatura (planejamento de produção), dentre outros (KASSAB et al., 2011; BANKS et al., 2008; MEDINA; CHWIF, 2007; FUJIMOTO, 2000).

Um Sistema Distribuído (SD) é um sistema formado por um conjunto de computadores (nós) independentes, que colaboram entre si e aparecem para seus usuários como um sistema único. O escalonamento de processos é responsável pela alocação e redistribuição dos processos, com o objetivo de balancear a carga e

alcançar um melhor desempenho. Um ambiente paralelo é uma coleção de ferramentas de *software* executando sobre um *cluster*, possibilitando a execução de aplicações distribuídas (TANENBAUM; STEEN, 2006).

Simular um modelo complexo de maneira sequencial, em um único processador, pode requerer muito recurso de processamento e levar muito tempo para ser executado. Esse fator justifica a utilização de ambientes paralelos para a execução de simulações. A simulação distribuída consiste na execução de um programa de simulação em um SD. A paralelização do modelo visa aumentar a velocidade de execução da simulação, objetivando simular sistemas complexos em menor tempo (CHWIF; PAUL; BARRETTO, 2006; FUJIMOTO, 2003).

Uma simulação sequencial necessita de algumas estruturas de dados para controlar a execução da simulação. Os eventos da simulação precisam ser executados em ordem, normalmente seguindo uma ordem cronológica, para evitar inconsistências conhecidas como erros de causa e efeito. Essa natureza sequencial de uma simulação faz com que as simulações distribuídas, além de necessitar das estruturas de dados de uma simulação sequencial, também necessitem de outras estruturas específicas para garantir a ordenação dos eventos. Essas estruturas específicas dependem da abordagem de sincronização de processos adotada, sendo a sincronização de processos o problema central das simulações distribuídas (CAROTHERS; FUJIMOTO, 2000).

Os protocolos de sincronização de processos são classificados como conservativos ou otimistas. Nos protocolos conservativos, a execução dos eventos só é permitida quando é certo que a ordenação dos eventos não será violada. Os protocolos otimistas não restringem a execução dos eventos. Partem da ideia otimista de que não haverá erros de ordenação. No entanto, esses erros podem ocorrer e, quando ocorrerem, os protocolos otimistas devem ser capazes de retornar a computação ao último ponto consistente, ou seja, aquele imediatamente anterior ao erro de ordenação. O *Time Warp*, proposto por Jefferson (1985), e o *Rollback* Solidário, proposto por Moreira (2005) são exemplos de protocolos otimistas.

São muitas as técnicas utilizadas para otimização do desempenho de simulações distribuídas. Fujimoto (2000) afirma que, dentre essas técnicas, as que mais se destacam são as técnicas de balanceamento de carga. Os algoritmos de balanceamento de carga desenvolvidos especificamente para simulação distribuída apresentam melhor desempenho que os desenvolvidos para aplicações paralelas tradicionais.

Os algoritmos de escalonamento estáticos baseiam-se em uma análise estática do modelo, realizam a alocação dos processos no início da execução do sistema e essa alocação inicial não é mais alterada até o fim da execução. O escalonamento dinâmico de processos realiza o balanceamento da carga em tempo real, ou seja, durante a execução do sistema e, através de migrações, distribui os processos na tentativa de equilibrar a carga. De Grande e Boukerche (2011) afirmam que o balanceamento estático é limitado a alguns tipos de aplicações, sendo incapaz de tratar as variações dinâmicas de carga presentes em muitas aplicações distribuídas.

1.1 Motivação

Diversos mecanismos de balanceamento de carga para simulação distribuída foram propostos, como, por exemplo, o algoritmo da utilização efetiva do processador (REIHER; JEFFERSON, 1990), o algoritmo da métrica do relógio local (BURDORF; MARTI, 1993) e o BGE (CAROTHERS; FUJIMOTO, 2000). Cada mecanismo apresenta suas regras específicas e se baseia em métricas, ou variações de uma mesma métrica, que estimam a carga do sistema e são utilizadas na tentativa de um melhor balanceamento da carga.

A motivação para a proposta deste trabalho vem do fato de que os estudos anteriores não avaliam as métricas individualmente, os objetivos eram avaliar os algoritmos como um todo. Nesses trabalhos, as métricas são avaliadas considerando diferentes algoritmos, modelos e fatores de desbalanceamento de carga, ou seja, não avaliam as métricas sob a ótica de um mesmo algoritmo de balanceamento, usando os mesmos modelos e com os mesmos fatores de desbalanceamento

de carga.

Considerando o desempenho superior dos protocolos otimistas em relação aos protocolos conservativos, o desempenho superior dos mecanismos de balanceamento dinâmicos em relação aos mecanismos de balanceamento estáticos e o fato de que o balanceamento de carga é a principal técnica de otimização para simulações distribuídas, esse trabalho realizou uma análise sobre as principais métricas para escalonamento dinâmico de processos em simulação distribuída usando o protocolo otimista *Time Warp*, o mais conhecido dentre os protocolos otimistas.

O estudo dos algoritmos de balanceamento de carga e dos fatores que contribuem para o desbalanceamento de uma simulação também motivou a proposta de uma nova métrica para balanceamento de carga para simulações distribuídas que utilizam protocolos otimistas.

1.2 Objetivos

O objetivo deste trabalho é avaliar as métricas de balanceamento de carga utilizadas pelos principais algoritmos de balanceamento de carga para simulações distribuídas que utilizam os protocolos otimistas, bem como propor uma nova métrica que poderá ser utilizada por algoritmos de balanceamento executando nesses ambientes.

Durante a pesquisa, foram realizadas as seguintes atividades:

- Implementação de um algoritmo de balanceamento de carga dinâmico para simulação distribuída que permite o uso de diferentes métricas. Desenvolvido com o propósito de avaliar as métricas sob a ótica de um único mecanismo de balanceamento;
- Desenvolvimento de uma nova métrica para índices de carga baseada nos diversos fatores que causam o desbalanceamento da carga, além de uma

avaliação dessa métrica juntamente com as outras métricas propostas anteriormente;

- Execução do algoritmo de balanceamento implementado para cada uma das métricas, considerando os mesmos modelos de simulação, as mesmas variações de carga e o mesmo ambiente de execução.

Com o intuito de analisar as métricas, este trabalho apresenta as especificações das mesmas, a forma como foram calculadas, o algoritmo de escalonamento desenvolvido, o ambiente onde as simulações foram executadas, os modelos de simulação utilizados e as variações de carga impostas.

1.3 Estrutura da dissertação

A dissertação foi estruturada de forma que a parte conceitual necessária para seu entendimento fosse apresentada no Capítulo 2. Esse capítulo faz uma introdução sobre SDs, ambientes paralelos, simulação e simulação distribuída. Apresenta também a classificação das simulações distribuídas e os principais mecanismos de sincronização de processos.

O Capítulo 3 aborda o escalonamento de processos e o balanceamento de carga. Apresenta os principais fatores do desbalanceamento de carga e os principais algoritmos de balanceamento de carga para simulações distribuídas.

As adaptações na implementação do protocolo *Time Warp* e no mecanismo de migração de processos, necessárias para a realização do trabalho, são apresentadas no Capítulo 4.

O Capítulo 5 apresenta as métricas avaliadas, a forma como elas foram calculadas e o algoritmo de balanceamento proposto para avaliá-las. Esse capítulo também introduz a nova métrica proposta.

As configurações dos parâmetros das simulações, o ambiente de execução, os

modelos de simulação propostos e as variações de carga impostas são detalhados no Capítulo 6.

O Capítulo 7 apresenta uma análise dos resultados obtidos e o Capítulo 8 apresenta as conclusões, contribuições e sugestões para trabalhos futuros.

2 Simulação e simulação distribuída

Este capítulo apresenta conceitos importantes para o entendimento de simulação e simulação distribuída.

2.1 Sistemas Distribuídos

Diversas definições sobre SDs são apresentadas na literatura, sendo, muitas vezes, discordantes entre si.

Lamport, Shostack e Paese (1982) definem um SD como aquele onde a falha de uma máquina, cuja existência é desconhecida pelo usuário, impede que o mesmo obtenha um determinado serviço solicitado.

Segundo Coulouris, Dollimore e Kindberg (2005), um SD é um sistema cujos componentes, de *hardware* ou *software*, localizados em uma rede de computadores, se comunicam e coordenam suas ações apenas através da troca de mensagens.

Para Tanenbaum e Steen (2006), um SD é um conjunto de computadores independentes, cooperando entre si e apresentando-se para os usuários de forma transparente, como se fosse um único sistema.

Já Belapurkar et al. (2009) definem um SD como aquele que envolve a interação entre entidades diferentes e independentes, se comunicando através de linguagens e protocolos estabelecidos e trabalhando em um objetivo comum.

Os SDs possuem diversas vantagens em relação aos sistemas centralizados,

tais como, a extensibilidade, capacidade de crescer gradativamente, o compartilhamento de recursos, a replicação, que aumenta a disponibilidade do sistema, e a mobilidade dada aos usuários, que podem acessar o sistema de diferentes lugares. Como desvantagens, podem ser destacadas as dificuldades no controle de acesso dos usuários, na manutenção da integridade dos dados replicados, no gerenciamento de acesso aos recursos, como por exemplo, alocação de tempo de CPU, na garantia de desempenho, que depende da arquitetura física da rede, e em relação à segurança das informações colocadas na rede (TANENBAUM; STEEN, 2006; BELAPURKAR et al., 2009).

2.2 Ambiente paralelo distribuído

Os SDs dão suporte à computação paralela por meio de *clusters* de computadores, também conhecidos como redes de estações de trabalho (*Network of Workstations* - NOW) . Um *cluster* é um conjunto de computadores conectados por uma rede de comunicação. Além de dar suporte às aplicações paralelas, os *clusters* são utilizados para outras finalidades, como, por exemplo, alta disponibilidade de aplicações (TANENBAUM; STEEN, 2006).

Pitanga (2003) classifica os *clusters* da seguinte forma:

- ***Cluster* de alto desempenho**: permite que computadores comuns possam lidar com uma grande carga de processamento;
- ***Cluster* de alta disponibilidade**: capaz de permanecer ativo por longos períodos de tempo;
- ***Cluster* para balanceamento de carga**: controla a distribuição de carga entre os nós.

Para Tanenbaum e Steen (2006), um ambiente paralelo distribuído é uma coleção de ferramentas de *software* executando sobre um *cluster*, criando uma camada

onde aplicações distribuídas podem ser executadas, possuindo as seguintes funções:

- **Gerenciamento de recursos**: conhecimento sobre a estrutura do *cluster*. Número de *hosts*, configuração dos *hosts* e informações de endereçamento, por exemplo;
- **Gerenciamento de processos**: gerenciamento das aplicações e processos em execução. Criação, destruição e alocação de processos;
- **Passagem de mensagens entre processos**: permite a comunicação entre os processos. Com o auxilio do ambiente, a troca de mensagens entre os processos é realizada de forma transparente.

O Capítulo 4 descreve o ambiente distribuído sobre o qual foram realizados os experimentos deste trabalho.

2.3 Simulação e simulação distribuída

Schriber e Brunner (2002) afirmam que uma simulação resulta da modelagem de um processo ou sistema, de modo que o modelo imite as respostas do sistema real em uma sequência de eventos.

Uma simulação consiste em avaliar um modelo matematicamente, onde os dados gerados são utilizados para estimar as características relevantes do modelo (LAW, 2007; CASSANDRAS; LAFORTUNE, 2008).

Através da simulação computacional, é possível modelar sistemas existentes e/ou conceituais. É a representação de um processo do mundo real sobre o tempo, com a geração de um histórico artificial do sistema que permite inferir sobre as características do sistema real (BANKS et al., 2008).

Para Law (2007), há problemas complexos do mundo real que só podem ser analisados através de simulações.

Em uma simulação computacional, o programa representa as mudanças de estados do sistema físico simulado. A simulação deve possuir algumas informações que possibilitem a representação do sistema físico, são elas: (1) uma estrutura que represente os estados do sistema, (2) uma forma de mostrar as mudanças de estados e a evolução do sistema e (3) alguma forma de representar o tempo.

O tempo de simulação segue algumas regras diferentes das regras de tempo físico. O tempo de simulação é um conjunto de valores ordenados (tempos lógicos), onde cada valor representa um instante do tempo físico do sistema que está sendo simulado. Essa ordenação do tempo segue as seguintes regras: seja L_1 o tempo lógico que representa o tempo físico F_1 e L_2 o tempo lógico que representa o tempo físico F_2. Se $L_1 < L_2$, então F_1 ocorreu antes de F_2 e L_1 ocorreu antes de L_2. Se $L_1 > L_2$, então F_1 ocorreu depois de F_2 e L_1 ocorreu depois de L_2. Essa natureza sequencial dos eventos de uma simulação sequencial é o grande problema enfrentado quando se deseja converter uma simulação sequencial em uma simulação distribuída.

Lamport (1978) observou que, em um sistema centralizado, o conhecimento do tempo real é irrelevante para a ordenação dos eventos. Sendo assim, concluiu que, em um SD, o importante é conhecer a ordem de execução dos eventos e não o tempo real de cada um deles. Propôs que um SD deve ser visto como uma sequência de eventos discretos, possibilitando a construção de uma relação de causalidade entre os eventos. Essa relação de precedência causal diz que, se um evento x precede um evento y, então, todos os processos devem concordar com o fato de que x ocorre antes de y. A relação de causa e efeito acontece se: (1) x e y são eventos do mesmo processo e se x ocorre antes de y ou (2) se x é o evento de envio de uma mensagem por um processo P_i e y é o evento de recebimento, em um processo P_j, da mensagem enviada pelo evento x. Um erro de causa e efeito ocorre quando a relação de precedência causal é violada.

A Figura 1 ilustra um erro de causa e efeito. A mensagem enviada pelo processo P_{01} ao processo P_{02}, com *timestamp* 3, viola a precedência causal, pois o processo

P_{02} já está no tempo lógico 4. Uma mensagem que viola a precedência causal é normalmente conhecida como mensagem *straggler*. O relógio local de um processo é conhecido como *Local Virtual Time* (LVT) e o relógio lógico global da simulação, que é o menor LVT dentre todos os LVTs dos processos da simulação, é conhecido como *Global Virtual Time* (GVT). Cada evento possui um *timestamp*, um valor que indica quando uma mudança deve acontecer no estado do sistema e os ordena cronologicamente.

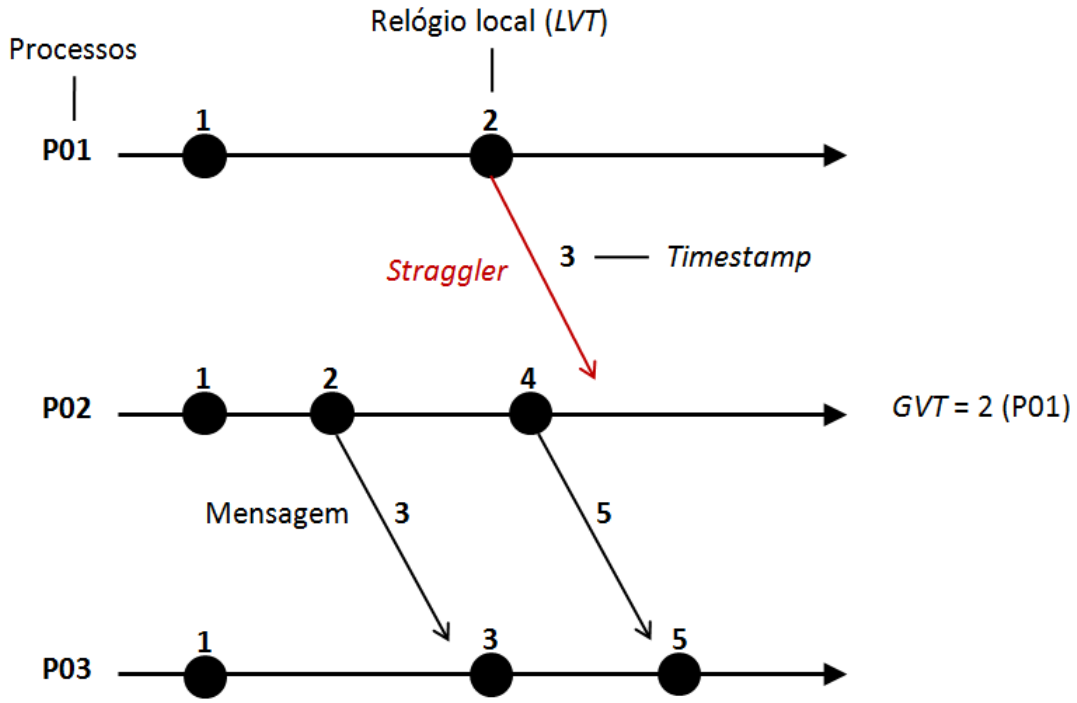

Figura 1: Violação da relação de causa e efeito.

Para projetar uma simulação, é necessário construir uma descrição do sistema a ser simulado. Essa descrição caracteriza um modelo, que contém as características representativas do sistema real. A simulação do modelo potencializa resultados mais precisos sem a necessidade de intervenção no sistema real.

Os modelos devem ser complexos o suficiente para responder as questões levantadas, mas não devem ser tão complexos quanto o sistema real. O modelo é uma

simplificação do sistema real, aproximando-se de seu comportamento. Se o modelo tiver uma complexidade maior que a do sistema real, passa a ser um problema, e não apenas um modelo (MEDINA; CHWIF, 2007).

Os modelos podem ser classificados como discretos ou contínuos. Os modelos discretos representam sistemas onde as mudanças de estado ocorrem em pontos específicos e não contínuos no tempo simulado. Um banco é um exemplo de um modelo discreto, onde o estado do sistema muda discretamente no tempo (número de clientes no banco, que é alterado quando um cliente entra ou sai do banco, por exemplo). Os modelos contínuos simulam sistemas onde as mudanças de estado ocorrem continuamente ao longo da simulação. Um avião em vôo é um exemplo de um modelo contínuo, com o estado do sistema mudando continuamente no tempo (posição e velocidade do avião, por exemplo) (LAW, 2007). Os modelos discretos são mais indicados para a representação de sistemas computacionais, devido ao comportamento discreto desses sistemas.

A utilização de SDs vem se tornando a forma mais viável para a execução de simulações distribuídas. Duas abordagens vêm sendo utilizadas, a *Multiple Replication in Parallel* (MRIP) e a *Single Replication in Parallel* (SRIP).

2.4 MRIP

A MRIP considera a paralelização de réplicas de um programa de simulação. O programa de simulação não precisa ser dividido em várias entidades de processamento. Ao invés disso, são utilizadas múltiplas instâncias do mesmo programa de simulação sequencial, porém, executando em paralelo, cada instância executando em um processador de forma independente (BAUSE; EICKHOFF, 2003).

As instâncias independentes enviam continuamente informações para um processo controlador, que faz uma análise global dos parâmetros da simulação. Através da análise dos parâmetros, o analisador global determina se há simulações suficientes para a precisão requerida para tomada de decisão e, quando essa precisão é

alcançada, a simulação pode ser encerrada.

As principais vantagens da abordagem MRIP são:

1. Pode ser aplicada a qualquer programa de simulação, sem a necessidade de alterá-lo;
2. Facilidade de uso e aplicação;
3. Devido à independência das simulações, se n simulações estão sendo executadas, então serão produzidos n resultados. O *speedup* é relativo ao número de processares;
4. Apresenta alta tolerância a falhas, pois a falha em uma simulação não afeta as outras. O ponto fraco é o analisador global que, em caso de falhas, compromete toda a simulação.

A abordagem MRIP só não se aplica quando o tamanho do modelo é grande e a replicação em um único processador levaria muito tempo para ser executada ou quando o resultado de cada replicação é semelhante, tornando a replicação ineficiente.

2.5 SRIP

A SRIP considera a paralelização de um mesmo programa de simulação. Divide o modelo de simulação e simula cada uma das partes em um processador diferente. A utilização de diversos processadores requer uma adaptação em relação à simulação sequencial. Dois pontos são identificados nessa adaptação: (1) o sincronismo entre os processos concorrentes e (2) as ferramentas usadas para implementar o paralelismo do programa (KUMAR, 2003). Nessa abordagem, o programa de simulação é dividido em vários processos. Os processos interagem uns com os outros exclusivamente pela troca de mensagens (eventos com *timestamps*).

Uma simulação sequencial necessita de algumas estruturas, que são as variáveis de estado, uma lista de eventos futuros e um relógio global. Os eventos precisam ser executados em sua ordem cronológica de *timestamps*, para evitar erros de causa e efeito. A simulação consiste basicamente em continuamente retirar e executar eventos da lista de eventos futuros. Uma simulação distribuída, além de necessitar das estruturas de uma simulação sequencial, também necessita de outras estruturas específicas para a simulação distribuída, que depende da abordagem de sincronização de processos adotada (FUJIMOTO, 2000).

2.5.1 Protocolos de sincronismo

Uma simulação sequencial se baseia em uma lista de eventos futuros, ordenada pelo tempo em que eles devem ser executados. Na abordagem SRIP, onde a simulação é dividida em diversos elementos de processamento, é necessária a utilização de um protocolo de sincronismo para garantir que a execução dos eventos siga a ordem cronológica.

Uma simulação executada de forma distribuída deve apresentar os mesmos resultados de quando executada de maneira sequencial. Isso só é possível através de mecanismos de sincronização, não necessários em uma simulação sequencial, pois os eventos já ocorrerem naturalmente ordenados. Sincronizar uma simulação distribuída consiste em evitar ou corrigir erros de causalidade no sistema sendo simulado. Fujimoto (2000) afirma que esse é um problema central na simulação distribuída e que os mecanismos de sincronização, classificados como conservativos ou otimistas, foram desenvolvidos visando tratar esse problema de ordenação da execução dos eventos.

Nos protocolos **conservativos**, a execução de um evento x só é permitida se não houver possibilidade da ocorrência de um novo evento y com o tempo cronológico anterior ao tempo cronológico de x, ou seja, impede a ocorrência de erros de causa e efeito. Já os protocolos **otimistas** não se preocupam se é seguro executar um determinado evento. Baseiam-se na idéia de que erros de causa e

efeito não irão acontecer, mas, caso ocorram, os protocolos otimistas devem ser capazes de retornar a computação ao último ponto consistente.

2.5.2 Protocolos conservativos

Os primeiros protocolos de sincronização criados se basearam na abordagem conservativa, que tem como principal função determinar quando é seguro processar um determinado evento (FUJIMOTO, 2000). Um processo da simulação só pode processar um evento da sua lista de eventos futuros se houver garantia de que a execução do evento não levará a ocorrência de erros de causa e efeito.

Um conceito importante para os protocolos conservativos é o *lookahead*. O *lookahead* é definido como sendo o intervalo de tempo em que um processo pode prever, com certeza, o que irá acontecer na simulação, sendo possível avançar até um determinado ponto sem a possibilidade de receber mensagens de outros processos com *timestamp* menor que o intervalo de duração do *lookahead*. Um *lookahead* l indica que o processo não receberá nenhuma mensagem com *timestamp* menor que $t + l$, sendo t o LVT do processo. O *lookahead* é o principal responsável pelo desempenho de uma simulação distribuída que utiliza protocolos de sincronização conservativos. Em geral, tenta-se buscar o maior *lookahead* possível para aumentar o desempenho da simulação (BRYANT, 1977; CHANDY; MISRA, 1979).

Um processo poderia determinar se é seguro executar o próximo evento da sua lista de eventos futuros observando o *timestamp* das últimas mensagens recebidas dos processos com os quais ele se comunica. Se o seu LVT for menor que o menor *timestamp* entre as últimas mensagens recebidas de cada processo, então é seguro executar o próximo evento de sua lista de eventos futuros.

Um problema comum, com o qual os protocolos conservativos precisam conviver, é a ocorrência de *deadlocks*, que podem ocorrer quando os processos esperam por uma possibilidade de ocorrência de erros de causa e efeito, ficando parados e não avançando na simulação.

O protocolo *Chandy Misra Bryant* (CMB), desenvolvido de maneira independente por BRYANT (1977) e CHANDY e MISRA (1979), é o mais conhecido dos protocolos conservativos e leva em seu nome uma homenagem aos seus criadores.

2.5.3 Protocolos Otimistas

Os protocolos otimistas não se previnem da ocorrência de erros de causalidade. Não impedem que os processos executem os eventos da lista de eventos futuros mesmo que isso leve a erros de causa e efeito. Na ocorrência de tais erros, os protocolos otimistas possuem mecanismos para corrigir os erros, permitindo que a simulação se mantenha consistente.

Nos protocolos otimistas, não é necessário que os processos enviem mensagens na ordem de seus *timestamps*, os canais de comunicação não precisam garantir a entrega das mensagens na mesma ordem de envio, processos podem ser criados ou encerrados durante a execução da simulação e também não é necessário definir, previamente, com quais processos um determinado processo pode se comunicar.

A maneira de restaurar a simulação para um estado consistente é conhecida como *rollback*. Um *rollback* retorna a computação a um ponto especifico consistente, simulando novamente os eventos afetados. Ao receber uma mensagem com *timestamp* menor que seu LVT, o processo restaura o estado salvo imediatamente anterior ao *timestamp* da mensagem, processa a mensagem recebida e prossegue com a simulação a partir desse ponto.

Os protocolos otimistas precisam tratar de dois fatores que podem comprometer o desempenho de uma simulação, que são a ocorrência de *rollbacks* e a necessidade de armazenar, continuamente, os estados de cada processo para possibilitar um possível *rollback*. A ocorrência de um *rollback*, além de fazer um processo perder o tempo gasto com a execução dos eventos desfeitos, o obriga a avisar os processos aos quais ele tenha enviado mensagens anteriores ao *rollback*, podendo causar *rollbacks* nesses processos. A necessidade de armazenar os estados

dos processos eleva o consumo da memória e requer que seja implementado um bom mecanismo para o seu gerenciamento.

A principal vantagem dos protocolos otimistas em relação aos protocolos conservativos é possibilitar uma melhor exploração do paralelismo, pois não bloqueiam processos em situações de possíveis erros de causalidade que acabam não acontecendo. O tratamento dos erros de causalidade permite que os protocolos otimistas explorem o paralelismo dos modelos, apresentando desempenhos superiores em relação aos protocolos conservativos (LAW, 2007).

O *Time Warp*, proposto por Jefferson (1985), é o protocolo otimista para sincronização de simulações distribuídas mais conhecido (MALIK; PARK; FUJIMOTO, 2009). Os conceitos e mecanismos fundamentais, tais como *rollback*, anti-mensagens e controle local e global do tempo de simulação, usados pelos protocolos otimistas, foram inicialmente introduzidos pelo *Time Warp*.

O *Rollback* Solidário (*Solidary Rollback*), proposto por (MOREIRA, 2005), é um outro protocolo otimista para sincronização de simulações distribuídas. O *Rollback* Solidário (*Solidary Rollback*) utiliza o conceito de *checkpoints* globais consistentes (BABAOGLU; MARZULLO, 1993) para identificar os pontos da simulação para onde os processos deverão retroceder em caso de *rollbacks*.

Esses protocolos se diferenciam na forma como realizam o procedimento para recuperar a computação a um estado consistente.

2.5.3.1 *Time Warp*

O protocolo *Time Warp* é constituído por duas partes distintas: (1) um controle local, interno a cada processo e responsável por garantir que os eventos do processo sejam executados em ordem cronológica, utilizando a política de escalonamento *Smallest Timestamp First* (STF) e (2) um controle global responsável por questões globais, tais como gerenciamento de memória e cálculo do GVT da simulação (JEFFERSON, 1985).

Cada processo no *Time Warp* possui as seguintes informações e características:

- **LVT**: relógio virtual local do processo;
- **Estado**: conjunto de variáveis que caracterizam o estado corrente do processo;
- **Lista de eventos futuros**: inclui todos os eventos que foram escalonados ao processo e ainda não foram processados. Os eventos escalonados podem ser resultantes de mensagens recebidas de outros processos da simulação e são ordenados pela ordem cronológica de seus *timestamps*;
- **Lista de mensagens negativas**: também conhecidas como anti-mensagens, são as mensagens enviadas a outros processos. Essa lista só é utilizada em caso de *rollbacks*;
- **Lista de estados**: os processos salvam cópias de seu estado a cada evento processado, para permitir a restauração em caso de *rollbacks*;
- **As mensagens carregam duas marcas de tempo**: o tempo virtual de envio, que é o LVT do processo emissor, e o tempo virtual de recebimento, que representa o tempo de simulação e é chamado de *timestamp*;

Duas situações podem levar um processo a realizar um *rollback*. A chegada de uma mensagem *straggler* ou a chegada de uma anti-mensagem.

Uma mensagem recebida por um processo, contendo um *timestamp* menor do que o seu LVT (*straggler*), leva o processo a realizar um *rollback* conhecido como ***rollback* primário**.

Quando um processo realiza um *rollback* primário, para cada mensagem enviada por ele com *timestamp* maior que o LVT de retorno, o processo deverá enviar uma mensagem indicando que ele realizou um *rollback*. Essas mensagens são as que ficam armazenadas na lista de mensagens negativas. Para cada mensagem existe

uma anti-mensagem. As duas mensagens são iguais, apenas com o sinal invertido. Uma mensagem possui sinal positivo e sua anti-mensagem correspondente possui sinal negativo.

Ao receber uma anti-mensagem, um processo executa uma das duas seguintes ações:

1. Se o evento correspondente à anti-mensagem ainda não foi processado, ou seja, ainda está em sua na lista de eventos futuros, o processo remove o evento da sua lista de eventos; ou

2. Se o evento correspondente à anti-mensagem já foi processado, o processo deve realizar um *rollback* para o estado imediatamente anterior ao *timestamp* da anti-mensagem. Esse tipo de *rollback* é conhecido como ***rollback* secundário**.

Em redes que não garantem a ordenação das mensagens, pode acontecer de uma anti-mensagem chegar antes de sua mensagem correspondente. Caso isso aconteça, o processo deve armazenar a anti-mensagem e, quando a mensagem correspondente chegar, eliminar ambas.

O mecanismo de controle global, usado em atividades como o gerenciamento da memória e como informação de entrada para algoritmos de balanceamento de carga, tem como ponto central o cálculo do GVT. Jefferson (1985) define o GVT no tempo real r como sendo o menor valor entre: (1) os LVTs de todos os processos da simulação no tempo r e (2) os *timestamps* de todas as mensagens em trânsito no tempo r. O GVT serve como limite inferior para os tempos virtuais, indicando que qualquer informação anterior a esse tempo pode ser descartada.

O algoritmo de Mattern (1993) é um dos mecanismos mais eficientes para o cálculo do GVT e será explicado a seguir por ter sido o mecanismo implementado no presente trabalho.

2.5.3.2 Algoritmo de *Mattern* para o cálculo do GVT

Esse mecanismo faz uso do conceito de corte consistente, dividindo a execução da simulação em partes, que permitem obter uma fotografia da simulação e, dessa forma, calcular o GVT.

Um corte é um subconjunto dos eventos executados pela simulação distribuída e o conjunto formado pelo último evento de cada processo contido no corte é conhecido como a fronteira do corte. Um corte é dito consistente se todas as mensagens que atravessam o corte têm sua origem à esquerda do corte. Dessa forma, um corte consistente é fechado à esquerda em relação à precedência causal.

Na Figura 2, o corte C_1, fronteira $P_{01}(3), P_{02}(2), P_{03}(3)$, é consistente, pois a única mensagem que passa pelo corte possui sua origem à esquerda do corte. O corte C_2, fronteira $P_{01}(3), P_{02}(2), P_{03}(4)$, não é consistente, pois umas das mensagens que passa pelo corte possui sua origem à direita do corte.

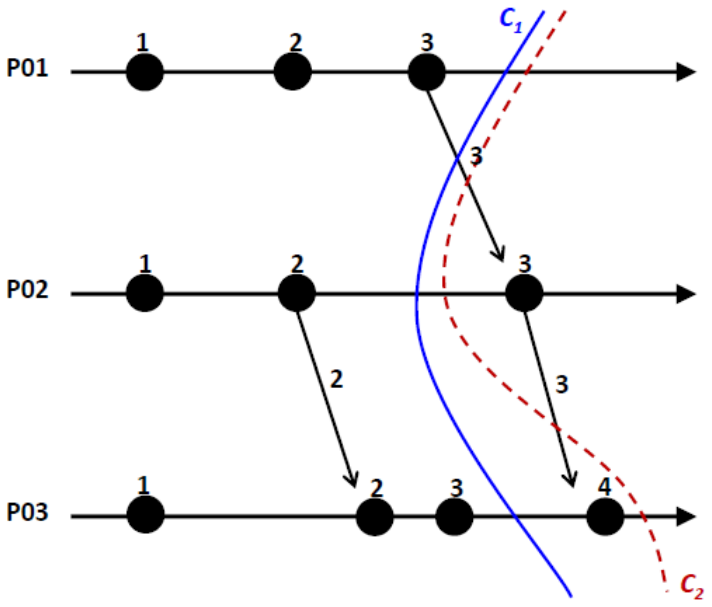

Figura 2: Exemplos de cortes. C_1 consistente e C_2 inconsistente.

A fronteira de um corte consistente inclui os estados locais dos processos e todas as mensagens transientes que passam pelo corte. O algoritmo de Mattern (1993) usa essas informações para realizar o cálculo do GVT. O algoritmo determina as mensagens transientes utilizando dois cortes e calcula o GVT na fronteira do segundo corte, como mostra a Figura 3.

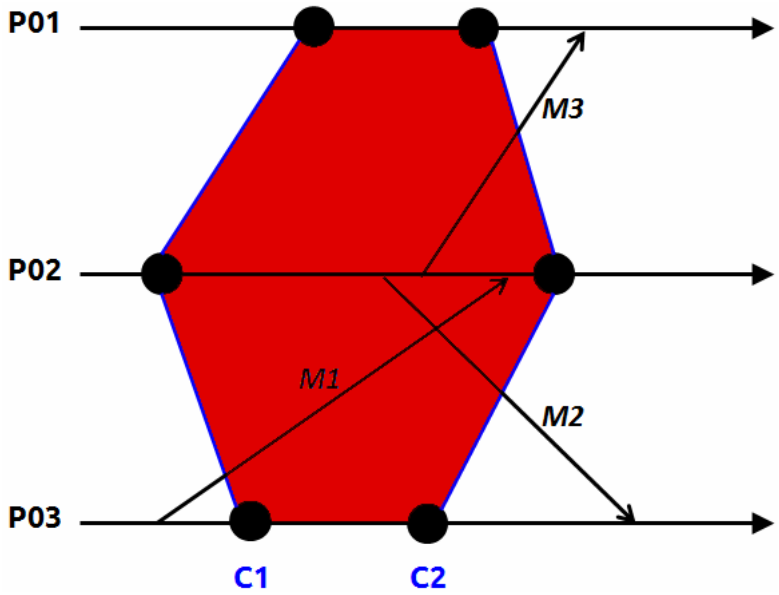

Figura 3: Cortes usados para determinar mensagens transientes.

O primeiro corte (C_1) solicita aos processos que iniciem o registro do menor *timestamp* das mensagens que enviarem. Essas mensagens podem ser transientes, no caso de atravessarem o segundo corte, e devem ser consideradas no cálculo do GVT. Tomando como base a Figura 3, os *timestamps* das mensagens *M2* e *M3* são considerados no cálculo do GVT. O segundo corte (C_2) deve garantir que todas as mensagens enviadas antes do primeiro corte já foram entregues aos destinatários, fazendo com que todas as mensagens que estão atravessando o segundo corte tenham sido enviadas após o primeiro corte, sendo assim, incluídas no cálculo do GVT. Pelo exemplo da Figura 3, o segundo corte só poderá ser realizado após o recebimento da mensagem *M1*.

O algoritmo usa um esquema de cores para determinar o momento em que cada processo está em relação aos dois cortes. Todos os processos iniciam com a cor branca, no primeiro corte são marcados com a cor vermelha e retornam a cor branca no segundo corte. As mensagens enviadas pelos processos carregam a cor que o processo estava quando as enviaram. Quando um processo está branco, envia mensagens brancas, já quando está vermelho, envia mensagens vermelhas.

Todas as mensagens brancas devem ser recebidas antes do segundo corte. Portanto, o segundo corte só poderá ser feito quando todas as mensagens brancas forem recebidas. Assim, todas as mensagens que passam pelo segundo corte serão vermelhas. O GVT é calculado como o menor valor entre: (1) os *timestamps* das mensagens vermelhas enviadas por todos os processos e (2) os LVTs de todos os processos da simulação. Ou seja, é o menor valor entre os LVTs dos processos e os *timestamps* das mensagens em trânsito.

2.6 Considerações Finais

Esse capítulo apresentou conceitos básicos sobre SDs, Simulação, Simulação Distribuída, com suas principais abordagens, e sobre os principais protocolos utilizados para sincronização de processos. Esses conceitos são importantes para a compreensão do restante deste trabalho.

Foram apresentadas as duas principais abordagens para Simulação Distribuída (MRIP e SRIP). A abordagem SRIP recebeu um foco maior, assim como os protocolos otimistas, em especial o *Time Warp*, pois formam a base para o ambiente desenvolvido neste trabalho.

O próximo capítulo apresenta uma revisão sobre escalonamento de processos e balanceamento de carga para Simulação Distribuída.

3 Escalonamento de Processos e Balanceamento de Carga em Simulações Distribuídas

Um processo é um programa em execução. Cada processo possui seu espaço de endereçamento, uma lista de posições de memória que o processo pode acessar. No espaço de endereçamento estão o código executável, os dados do programa e a pilha de execução. O processo também armazena sua atividade corrente, representada pelo contador de programa e pelo conteúdo de seus registradores. Em sua pilha de execução ficam armazenados dados temporários, como por exemplo, parâmetros de funções, endereços de retorno e variáveis locais. Na seção de dados ficam armazenadas as variáveis globais. Um processo também possui outros recursos, tais como, uma lista de arquivos abertos, uma lista de processos relacionados, um *heap* de memória, que é a memória dinamicamente alocada durante sua execução, e outras informações necessárias para a execução do programa (TANENBAUM, 2010; SILBERSCHTZ; GALVIN; GAGNE, 2009).

O escalonamento de processos é a atividade responsável pela alocação dos processos aos processadores, com o objetivo de atender as demandas específicas e/ou manter a carga do sistema equilibrada. É importante diferenciar bem quando o escalonamento é realizado em um ambiente com um único processador e quando é realizado em um ambiente com diversos processadores (várias estações de trabalho). Dois níveis de escalonamento são definidos de acordo com o ambiente e tipo da aplicação.

Tanenbaum (2010) apresenta um nível onde o escalonador de processos é responsável por monitorar a execução dos processos no sistema operacional, cuidando de questões como: a ordem de execução dos processos, em qual processador cada processo será alocado (em caso de arquiteturas com mais de um processador) e a fatia de tempo (*Time Slice*) que cada processo terá para executar no processador.

Um segundo nível, muitas vezes chamado de escalonamento global de processos, é responsável por determinar qual estação de trabalho do SD é a mais adequada para a execução de um determinado processo. Nesse nível, os objetivos da aplicação e a plataforma como um todo são considerados, para que se consiga uma melhor alocação dos processos (TANENBAUM; STEEN, 2006; SOUZA, 2000; CASAVANT; KUHL, 1988).

Geralmente, algoritmos de escalonamento de processos são desenvolvidos para ambientes paralelos onde são executados programas paralelos, esses compostos por muitos e, possivelmente, diferentes processos. A decisão sobre a alocação dos processos é tomada com base em informações sobre a plataforma, o *hardware*, as aplicações e a carga do sistema. Um programa de simulação distribuída, além de ser afetado por todos os fatores citados, também sofre desbalanceamento causado pelas características do próprio modelo da simulação.

3.1 Escalonamento de processos em sistemas distribuídos

Balancear a carga em um SD consiste em alocar os processos aos processadores de forma que as cargas desses processadores estejam equilibradas. Essa alocação pode ser feita apenas no início da execução da aplicação ou através da migração de processos entre os processadores. Os objetivos dos algoritmos de distribuição de carga são: (1) a maximização da velocidade de execução das aplicações e (2) a busca pelo nivelamento da carga entre os diversos processadores.

A carga de um sistema é medida através de índices de carga. Um índice de carga mede a utilização de algum elemento do sistema (BRANCO, 2004). Diversos

índices de carga podem ser usados para o balanceamento da carga de um sistema. A escolha do(s) melhor(es) índice(s) a ser(em) usado(s) depende das características da aplicação. A utilização do processador, o tamanho da fila de processamento e a quantidade de memória disponível são alguns exemplos de recursos utilizados para se calcular índices de carga.

Cada tipo de aplicação paralela possui suas particularidades. Com a simulação distribuída não é diferente e muitos algoritmos foram desenvolvidos voltados especificamente para o balanceamento de carga desse tipo de aplicação. Os algoritmos desenvolvidos especificamente para simulação distribuída, utilizando informações sobre parâmetros da simulação, apresentam melhores desempenhos que aqueles desenvolvidos para aplicações paralelas em um âmbito geral (VOORSLUYS, 2006).

Os problemas de balanceamento de carga se tornam ainda mais complexos em simulações distribuídas executando sobre protocolos otimistas. Nesses ambientes, as simulações são executadas com o objetivo de completar a simulação o mais rápido possível, os mecanismos de balanceamento de carga trabalham para atender a esse propósito, ao invés de simplesmente tentar maximizar a utilização dos recursos, uma vez que muito do trabalho realizado poderá ser desfeito no futuro devido aos *rollbacks*. Apenas fazer com que todos os processadores operem com a mesma carga não garante que uma simulação distribuída esteja equilibrada. Em geral, os algoritmos de escalonamento para simulação distribuída otimista operam com o intuito de reduzir ao máximo a ocorrência de erros de causa e efeito.

3.2 Classificação dos tipos de balanceamento para simulações distribuídas

Carvalho Junior (2008) propôs uma classificação para os algoritmos de balanceamento de carga, especificamente os que executam sobre protocolos otimistas, baseada no momento em que os algoritmos realizam suas atividades de monitoramento, avaliação e alocação dos processos. Essa classificação separa os algoritmos

em três grupos, **balanceamento em tempo de execução**, **balanceamento em tempo de mapeamento** e **balanceamento misto**.

Os algoritmos que fazem o **balanceamento em tempo de mapeamento** fazem um mapeamento dos dados antes que a execução da simulação inicie. Baseado na análise dos dados do modelo da simulação, dados coletados em execuções anteriores ou dados de avaliação da plataforma, tomam as decisões de como realizar a alocação dos processos aos processadores e essa alocação inicial não é mais alterada. Essa característica de comportamento faz com que esses mecanismos sejam classificados como de **balanceamento estático**.

Os algoritmos que fazem o **balanceamento em tempo de execução** monitoram informações específicas da simulação e/ou da carga do sistema em tempo de execução e tomam decisões de realocação dos processos durante a execução da simulação. Alguns autores classificam esses algoritmos como de **balanceamento dinâmico**. Esses algoritmos fazem uso de mecanismos de migração de processos para poder deslocar processos entre os nós da aplicação.

Os **algoritmos de balanceamento misto** realizam tanto uma análise no modelo e plataforma para realizar a alocação inicial dos processos quanto monitoram os dados da simulação e/ou ambiente para, caso necessário, efetuar a realocação de processos.

O balanceamento de carga é um componente importante para melhorar o desempenho das aplicações que executam em ambientes distribuídos. Muitas técnicas de balanceamento dinâmico foram desenvolvidas até chegar a um desempenho bom o suficiente para tornar o balanceamento dinâmico viável. O balanceamento estático torna-se inviável devido à variação no comportamento dos processos e da carga de cada processador durante a execução (DE GRANDE; BOUKERCHE, 2011).

3.3 Migração de processos

A migração de processos é necessária para permitir que os algoritmos de escalonamento equilibrem a carga do sistema, pois permite modificar a distribuição dos processos entre os computadores dinamicamente.

Os objetivos da migração de processos são: (1) o balanceamento dinâmico da carga, (2) aproximar os processos dos recursos que acessam, (3) permitir tolerância a falhas, (4) melhoria no desempenho da comunicação entre os processos e (5) a administração do sistema (MILOJICIC et al., 2000).

Migrar um processo em execução significa transferir seu estado entre dois computadores de um SD, permitindo que a execução do processo possa continuar no computador de destino do mesmo ponto em que foi suspensa no computador de origem. Assim que for determinado o processo que sofrerá migração, o mecanismo de migração de processos é responsável por suspender a execução do processo no computador de origem, enviar o estado do processo para o computador de destino e reiniciar a execução do processo.

Um bom mecanismo de migração de processos é fundamental para o trabalho dos algoritmos de balanceamento de carga dinâmicos. Mais detalhes sobre a migração de processos podem ser obtidos em (JUNQUEIRA, 2012) e (RAVASI, 2009).

3.4 Fatores que causam desbalanceamento da carga em simulações distribuídas

Carothers e Fujimoto (2000) afirmam que simulações distribuídas, executando sobre protocolos otimistas, estão sujeitas a diversos fatores que afetam o balanceamento de carga. Esses fatores podem ser internos ou externos à simulação. São três os fatores internos que podem causar desbalanceamento da carga, a **população de eventos**, a **granularidade dos eventos** e a **comunicação**. São dois os fatores externos, a **carga dos usuários** e a **heterogeneidade dos processadores**. Os

fatores externos se manifestam quando as simulações estão sendo executadas em ambientes não dedicados à simulação, onde uma carga externa ao sistema provida por usuários do ambiente pode existir e variar no tempo.

É importante ressaltar que é possível ter diversas combinações desses fatores. Determinar ou simular todos esses fatores e suas possíveis combinações não é uma tarefa simples.

Se a população de eventos estiver irregular, o número de eventos processados em um determinado período de tempo pode ser diferente entre os processos da simulação. Processos executando um grande número de eventos, com um valor muito superior em relação a outros processos, podem levar a situações de alta carga em seus respectivos processadores. Por outro lado, processos com um número menor de eventos podem levar a situações de baixa carga em seus respectivos processadores. Estas situações de desbalanceamento farão processos alocados em processadores com baixa carga sofrer *rollbacks* causados por processos alocados em processadores com alta carga. Um caso especial de irregularidade na população de eventos é quando todos os processos da simulação possuem uma quantidade similar de eventos, mas o número de processos alocados em cada processador difere muito.

Uma granularidade diferente dos eventos, é causada quando o montante de tempo gasto para processar os eventos varia significativamente entre os processos. É assumido que o tempo de CPU gasto para o salvamento de estado faz parte dos custos da execução dos eventos. Dessa forma, mesmo se a população de eventos for semelhante entre os processos, os processos com eventos que levam mais tempo para executar irão sobrecarregar seus processadores, enquanto processos que levam menos tempo farão com que seus processadores tenham uma carga menor. Esta situação de desbalanceamento também fará com que processos alocados em processadores com baixa carga sofram *rollbacks* causados por processos alocados em processadores com alta carga.

O desbalanceamento de carga, devido à comunicação entre os processos, está relacionado ao tempo de CPU gasto para receber e enviar mensagens e ao custo de

comunicação (em termos de latência). A variação no padrão de troca de mensagens entre os processos pode levar a situações de desbalanceamento de carga. Processadores com gastos de CPU elevados devido às tarefas de comunicação dos seus processos causarão *rollbacks* em processos alocados em processadores com menor carga de comunicação. Carothers e Fujimoto (2000) evidenciam a dificuldade em simular esse tipo de desbalanceamento.

Em um ambiente não dedicado exclusivamente à simulação, usuários externos podem executar diversos programas que irão competir com os processos da simulação por recursos do ambiente, como tempo de CPU. Esta situação pode levar a sobrecarga de alguns processadores e consequentemente a um desbalanceamento na carga. Para a simulação, significa que processos alocados em processadores sobrecarregados terão sua taxa de avanço reduzida, o que levará a situações de *rollback*.

A heterogeneidade dos processadores é um outro fator de desbalanceamento de carga. Ocorre em ambientes onde as características dos processadores variam entre os nós. Processos executando em processadores com menor poder de processamento tendem a ter uma taxa de avanço menor do que processos executando em processadores com maior capacidade de processamento, fazendo com que haja um desbalanceamento da carga. Em um ambiente heterogêneo, processadores com menor capacidade de processamento podem ser comparados com processadores sobrecarregados pertencentes a um ambiente homogêneo.

3.5 Algoritmos de escalonamento para simulação distribuída

Os algoritmos de balanceamento de carga específicos para simulação distribuída, por meio da utilização de informações da própria simulação, procuram controlar os fatores que causam o desbalanceamento da carga da simulação. No caso de simulações distribuídas usando protocolos otimistas, os algoritmos de balance-

amento visam a redução do número de *rollbacks*.

3.5.1 Algoritmo da utilização efetiva do processador

Para Reiher e Jefferson (1990), o balanceamento dinâmico de carga tem os objetivos de equilibrar o desempenho da simulação, tratar efetivamente a criação e destruição de processos e eliminar a necessidade de intervenção dos usuários no escalonamento dos processos. Acreditam que as técnicas de balanceamento de carga baseadas na utilização do processador não são aplicáveis às simulações otimistas. Propuseram uma técnica que realiza o balanceamento de carga através de uma métrica mais generalizada, chamada de *utilização efetiva do processador*, de agora em diante citada como *utilização efetiva*.

Em uma simulação otimista, o trabalho executado pode ou não sofrer *rollbacks*. O trabalho executado que não sofreu *rollback* é chamado de *trabalho efetivo*. A fração de *trabalho efetivo* realizado, em relação a todo trabalho realizado por um processador, é a *utilização efetiva*. Um processador com alta *utilização efetiva* está contribuindo muito com a simulação, enquanto um processador com baixa *utilização efetiva* está contribuindo menos. A política baseia-se em equilibrar a *utilização efetiva* de todos os processadores da simulação. O escalonador deve tentar transferir carga de processadores realizando muito *trabalho efetivo* para processadores realizando pouco *trabalho efetivo*.

O algoritmo calcula a *utilização efetiva* dos processos e também dos processadores. A *utilização efetiva* de um determinado processador é a soma das utilizações dos processos alocados nesse processador. Quando há desbalanceamento na carga, processos são migrados dos processadores mais carregados, maior *utilização efetiva*, para processadores menos carregados, menor *utilização efetiva*. O processador mais sobrecarregado faz par com o processador menos sobrecarregado. O segundo processador mais sobrecarregado com o segundo menos sobrecarregado e assim sucessivamente. Apenas pares de processadores onde as *utilizações efetivas* diferem em um determinado valor participam da migração de processos. Esse valor de di-

ferença deve ser superado para que haja migração. Ele determina o quão sensitivo será o mecanismo de migração. Valores baixos irão causar migrações desnecessárias e valores altos farão com que migrações necessárias não sejam realizadas.

A migração objetiva igualar o tempo de processamento que cada processo recebe, para que os processos tenham taxas de progresso próximas. Assim, processos com baixa utilização são migrados para processadores mais carregados e recebem menos tempo de processamento, já que baixa utilização significa muito tempo de processamento em relação a outros processos e, consequentemente, muitos *rollbacks*. Processos com alta utilização são migrados para processadores mais ociosos e recebem mais tempo de processamento, já que alta utilização significa pouco tempo de processamento e consequentemente poucos *rollbacks*, mas, no entanto, causam muitos *rollbacks* em outros processos.

O mecanismo calcula, dinamicamente, o *trabalho efetivo* para realizar o balanceamento da carga. Uma determinada porção de trabalho só poderá ser contabilizada como *trabalho efetivo* quando não puder mais sofrer *rollbacks*, o que faz com que a *utilização efetiva* só possa ser calculada levando em consideração os eventos que não possam mais sofrer *rollbacks*.

A Equação 3.1 exibe a primeira forma de calcular a *utilização efetiva* de um processo.

$$utilização\ efetiva = \frac{número\ de\ eventos\ efetivados}{número\ de\ eventos\ processados} \tag{3.1}$$

Essa forma de calcular a *utilização efetiva* possui uma deficiência, pois apenas podem ser considerados como eventos efetivados aqueles cujo LVT são menores ou iguais ao GVT, pois não se pode garantir que eventos com LVT maiores que o GVT não sofrerão *rollbacks*.

Para resolver esse problema, deve ser usada uma *estimativa para o trabalho efetivo*. Essa estimativa é baseada em contabilizar quantos ciclos de processador foi usado por cada evento. Esse valor deve ser adicionado à *estimativa de trabalho efetivo* do processo. Se houver *rollbacks*, o trabalho dispensado para realizá-los,

ciclos de processador, deve ser subtraído da *estimativa do trabalho efetivo*. Já que a *utilização efetiva* é a fração de tempo gasto em *trabalho efetivo* e sendo o *trabalho efetivo* uma estimativa, a *utilização efetiva*, quando calculada dessa forma, também será uma estimativa. Essa estimativa pode estar errada e resultar em migrações desnecessárias em alguns momentos.

Nessa abordagem, o cálculo da *utilização efetiva* de um processo é feito de acordo com Equação 3.2.

$$utilização\ efetiva = \frac{total\ de\ ciclos\ de\ processamento\ -\ total\ de\ ciclos\ perdidos}{total\ de\ ciclos\ de\ processamento} \tag{3.2}$$

Uma possível métrica para o balanceamento é fazer com que os LVTs de todos os processadores progridam uniformemente, medindo a carga pelo LVT. LVT muito alto, em relação aos outros processadores, significa baixa carga. LVT baixo significa alta carga. Segundo os autores, o balanceamento baseado na *utilização efetiva* é melhor do que o balanceamento baseado na taxa de progresso dos LVTs, pois é invariante em relação à ordem dos LVTs. O problema em considerar o LVT como métrica é que nem todos os eventos, que incrementam o LVT em um determinado valor, levam o mesmo tempo de processador para ser executado.

Como exemplo, consideremos que, para ser executado, o evento de LVT 100 gaste menos processamento em relação ao evento de LVT 200. Do ponto de vista do tempo virtual, o tratamento de ambos os eventos faz com que o LVT avance em n unidades. No entanto, se o objetivo do balanceamento de carga for aproximar a taxa de progresso dos LVTs o mesmo tratamento deveria ser realizado em ambos os casos. Porém, como há mais processamento gasto no evento de LVT 200, o balanceamento de carga deveria dedicar mais esforços em garantir que essa parte da simulação execute adequadamente, deveria haver mais migrações nessa etapa do que no evento de LVT 100, por exemplo. Na realidade, o que deve ser balanceado é a porção de ciclos de processador úteis sendo gastos e não a propagação dos LVTs.

Considere duas aplicações iguais, exceto pelo fato de que cada evento na se-

gunda aplicação tem LVT diferente ao seu evento correspondente da primeira aplicação. São os mesmos eventos gerando as mesmas instruções na mesma ordem, apenas os LVTs são diferentes, não uniformemente diferentes. Um conjunto de migrações suficiente para a primeira aplicação deveria ser bom para a segunda. No entanto, se a decisão de migrar for baseada na taxa de progresso dos LVTs, uma variação nos LVTs da segunda aplicação levará a diferentes conjuntos de migração. Em uma política baseada em *utilização efetiva*, considerando que as mesmas instruções sejam realizadas, a *utilização efetiva* será a mesma e o mesmo conjunto de migração seria suficiente.

Os resultados obtidos nos testes realizados por Reiher e Jefferson (1990) apontam para um aumento de desempenho de, em média, 13% a 19% quando usado o algoritmo de balanceamento proposto. Para algumas configurações há ainda melhor desempenho, no entanto, para algumas configurações houve uma queda no desempenho. Diversos testes foram realizados, inclusive variando o parâmetro que determina quando haverá migração entre dois processadores que foram considerados, respectivamente, o mais e o menos carregado. Os resultados mostram que, para um valor muito alto, os resultados não são bons, pois não houve migrações sugestivas, e que, geralmente, valores mais baixos são mais adequados, podendo variar de acordo com outras configurações, como por exemplo, o número de processadores.

3.5.2 Algoritmo da métrica do relógio local

Nessa técnica, proposta por Burdorf e Marti (1993), o escalonador de processos determina qual o processador mais ocioso e então migra processos para esse processador. Além dos processos da simulação, considera outros processos de usuários no cálculo da carga do sistema. Usa tanto o balanceamento estático quanto o balanceamento dinâmico.

O balanceamento **estático** determina como será a atribuição inicial dos processos aos processadores. Verifica o peso de cada processador antes de atribuir

processos aos mesmos. Os pesos dos processadores são estabelecidos de acordo com o tempo relativo de execução de uma simulação sequencial previamente executada. Os processos também têm pesos associados a eles. O peso de cada processo pode ser determinado pelo programador da simulação ou ajustados com um valor padrão. Em tempo de carga, os pesos dos processos são combinados para determinar a melhor distribuição dos processos aos processadores. Uma vez feita a atribuição inicial o balanceamento estático termina seu trabalho e não tem mais nenhuma influência na simulação. Burdorf e Marti (1993) afirmam que esse modelo de alocação estático é 10% mais efetivo que o modelo *Round Robin*, pois cada processador recebe uma carga compatível com sua capacidade.

Para minimizar o número de *rollbacks*, o balanceamento dinâmico procura diminuir a variação dos LVTs entre todos os processos da simulação.

Seja $M = \{m_1, ..., m_n\}$ o conjunto de todos os processadores da simulação, T o tempo de simulação de um determinado processo e $X = \{x_1, ..., x_n\}$ o conjunto com o menor LVT entre os processos alocados em cada processador. Calcula-se então a média, $\bar{x}$, e o desvio padrão, s, dos valores em X. Não devem ser considerados valores em X que tendem ao infinito, pois esses inviabilizariam o uso do desvio padrão calculado.

A média dos valores em X é calculada pela Equação 3.3.

$$\bar{x} = \frac{\sum_{i=1}^{n} LVT_i}{n} \tag{3.3}$$

As Equações 3.4, 3.5 e 3.6 mostram como o desvio padrão é calculado.

$$s_i = LVT_i - \bar{x} \tag{3.4}$$

$$variância = \frac{\sum_{i=1}^{n} (s_i)^2}{n} \tag{3.5}$$

$$s = \sqrt{variância} \tag{3.6}$$

Calculados a média e o desvio padrão, δ deve ser ajustado subtraindo a média

pelo desvio padrão, conforme mostra a Equação 3.7.

$$\delta = \bar{x} - s \tag{3.7}$$

Se existir um $T(x_j) < \delta$, move-se o processo mais atrasado, x_j, para o processador mais adiantado m_k, onde m_k é representado pelo maior LVT em X.

Para evitar o problema da flutuação (BARAK; SHILOH, 1985), que mostra uma flutuação da carga logo após um processo de migração, podendo levar a informações enganosas, o escalonador espera que o GVT seja calculado 3 vezes depois da última migração antes de analisar a carga novamente. Nesse momento, o sistema já está em equilíbrio e a carga pode ser analisada com mais segurança.

Essa abordagem pode falhar se for considerado um ambiente onde um processador está bem à frente e os demais possuem valores menores e próximos, não havendo migração de processos. Para resolver esse problema, γ deve ser calculado como a soma da média com o desvio padrão, Equação 3.8.

$$\gamma = \bar{x} + s \tag{3.8}$$

Se existir um $T(x_k) > \gamma$, move-se o processo x_j, o processo mais atrasado do processador mais atrasado, para m_k, onde m_k é representado pelo maior LVT em X. Assim, o processo mais atrasado do processador mais atrasado é migrado para o processador onde se encontra o processo mais adiantado, pois ele é um dos processos que tem mais trabalho a fazer.

Os resultados apresentados por Burdorf e Marti (1993) mostram que, baseado nas simulações executadas, o algoritmo proposto tem melhores resultados que o algoritmo da *utilização efetiva* proposto por Reiher e Jefferson (1990), quando calculada baseada no número de mensagens que sofreram *rollbacks* pelo número de mensagens processadas.

Um outro algoritmo que também trabalha na tentativa de minimizar a dife-

rença entre os LVTs dos processos da simulação é o algoritmo apresentado por Glazer e Tropper (1993). Os autores definiram a *taxa de avanço da simulação* como sendo a taxa de avanço do tempo de simulação de um processo em função do tempo de CPU dado a ele. Essa taxa depende do tempo gasto para processar um evento, do quanto o LVT avança por evento processado e da frequência de eventos. A diferença entre os LVTs dos processos é minimizada controlando a quantidade de tempo de processamento destinado a cada processo. Processos mais lentos recebem, proporcionalmente, mais tempo de processador. Por exemplo, se um processo P tem o dobro da taxa de avanço de simulação em relação a um processo Q, pode ser atribuído a P a metade do tempo de CPU atribuída a Q. Pode ser necessário migrar processos para outros processadores caso o ajuste do tempo de processamento dos processos resulte em um desequilíbrio na carga do processador.

3.5.3 Algoritmo probabilístico

A ideia básica, proposta por Som e Sargent (1998), é estimar, explicitamente, a probabilidade de que um evento, uma vez processado, sofra *rollback* causado devido ao recebimento de uma mensagem *straggler*. O método usa algoritmos probabilísticos para determinar a sequência de eventos a ser executada em um determinado processador.

Sendo L_1 e L_2 processos alocados em um processador P. A fila de entrada de L_1 contém um evento processado $a(5)$, onde 5 é o *timestamp* do evento, e dois eventos não processados $b(9)$ e $c(11)$. A fila de entrada de L_2 contém um evento processado $x(4)$ e dois eventos não processados $y(10)$ e $z(14)$. Suponha que x tenha sido processado anteriormente e que a acabou de ser processado. Sendo o menor *timestamp* considerado como mecanismo de escalonamento, tanto b quanto y são candidatos a ser o próximo evento a ser executado.

Na abordagem probabilística são utilizadas distribuições aleatórias baseadas em informações de eventos passados para determinar a probabilidade de uma men-

sagem, com *timestamp* menor que o de b, não ser recebida no futuro. Uma estimativa similar é feita para y. O evento com a maior probabilidade, de não sofrer *rollback*, é selecionado como o próximo evento a ser executado por P. O evento com menor probabilidade não é, necessariamente, o evento de menor *timestamp*. O algoritmo efetua essa previsão em vários níveis, determinando a sequência de eventos mais adequada para ser executada.

O algoritmo mais comum para ordenar a lista de eventos a ser executadas é o STF, que ordena os eventos do menor para o maior *timestamp*, considerando que eventos com *timestamp* menores são menos propensos a sofrer *rollbacks* por uma mensagem *straggler*. Comparando com o STF, a abordagem proposta por Som e Sargent (1998) reduz a ocorrência de *rollbacks* em porcentagens que variam de 2% a 42%, essa variação é devida ao uso de diferentes modelos, com diferentes padrões de comunicação, incremento de *timestamps* e granularidade dos eventos (tempo de CPU gasto para processar os eventos).

3.5.4 SGs - *Strong Groups*

Som e Sargent (2000) utilizaram-se do conceito de *utilização efetiva*, identificando algumas fraquezas na abordagem apresentada por Reiher e Jefferson (1990). Consideraram a estrutura do modelo para realizar o escalonamento dos processos. A simulação é vista como um conjunto de SGs. Cada SG é uma coleção de processos, onde todos os processos pertencentes a um mesmo SG têm uma forte influência na taxa de progresso uns dos outros.

Os SGs são identificados analisando o grafo de interconexão criado a partir do modelo da simulação. Para cada modelo de simulação, pode-se construir um grafo direcionado, onde cada nó do grafo representa um processo do modelo. Uma aresta do processador L_i para o processador L_j indica que L_i pode enviar uma mensagem para L_j.

A Figura 4, (a) e (b), mostram o modelo e o grafo direcionado respectivamente.

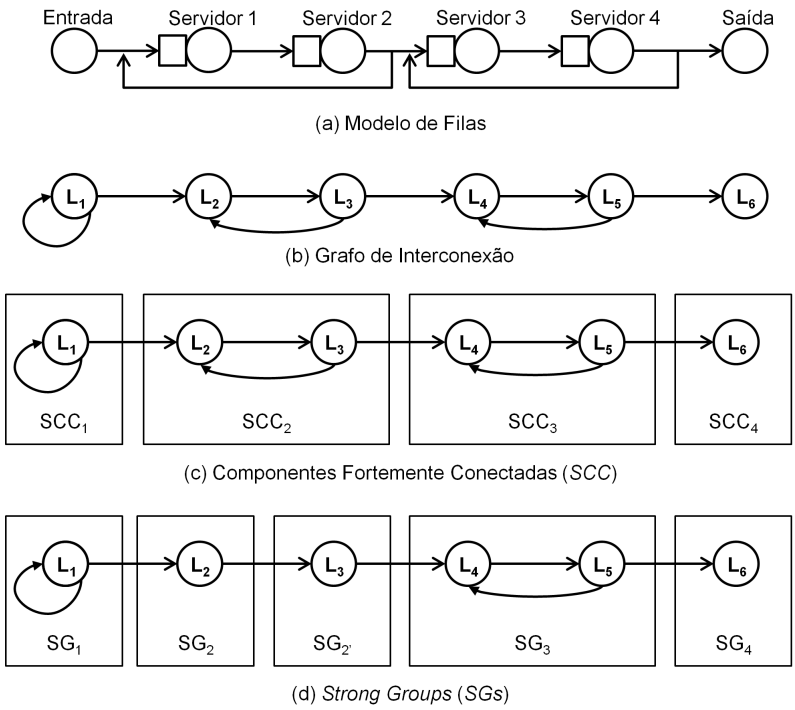

Figura 4: Modelo de fila, grafo de interconexão, SCCs e SGs (SOM; SARGENT, 2000).

Um processo L_i pode afetar, diretamente, outro processo L_j, se existir uma aresta de L_i para L_j ou pode afetar, indiretamente, se houver um caminho entre eles no grafo direcionado. De acordo com a Figura 4 (b), L_1 pode afetar, diretamente, L_2 e, indiretamente, L_3.

Um grafo direcionado pode ter uma ou mais componentes fortemente conectadas, *Strongly Connected Components* (SCC) (Figura 4 (c)). Em um grafo direcionado, uma SCC é um conjunto de vértices tal que, para cada par de vértices u e v, há um caminho de u para v e um caminho de v para u.

É possível que um processo L_i raramente envie mensagens para um processo L_j. Nesse caso, a aresta que liga L_i a L_j, no grafo direcionado, é considerada um *link* fraco. As arestas que são mais frequentemente usadas são consideradas *links* fortes. Todos os *links* fracos do grafo direcionado podem ser removidos. Por exemplo, se o *link* entre os processos L_3 e L_2 for considerado fraco, ele pode

ser removido. Essa remoção quebra a SCC SCC_2 nos SGs SG_2 e $SG_{2'}$. Após a remoção de todos os *links* fracos, as SCCs resultantes são os SGs, Figura 4 (d).

O algoritmo assume que processos dentro da mesma SCC não podem progredir muito a frente uns dos outros. A diferença na taxa de progresso depende de quão frequentemente os processos de uma SCC interagem uns com os outros, estando diretamente ligado ao número de *links* fracos da SCC em questão. SCCs com muitos *links* fracos levam a um limite alto e vice-versa. Considerando que um SG é uma SCC sem *links* fracos, conclui-se que os progressos dos processos de um mesmo SG permanecem próximos uns dos outros. Assim, os SGs podem ser vistos como uma única entidade.

No entanto, diferentes SGs não são restringidos, por propriedades estruturais, a progredirem no mesmo ritmo e essa taxa desigual de progresso entre os SGs pode causar:

1. Problemas de balanceamento;
2. Longo período de tempo para completar a execução;
3. *Rollbacks* em cadeia.

Quando dois processos, um mais rápido e outro mais lento, pertencem a SGs diferentes, duas situações podem ocorrer: não haver *rollbacks* ou, quando ocorrerem, os *rollbacks* serão tardios. Enquanto não houver *rollbacks*, haverá possíveis problemas de sobrecarga e ociosidade. Quando houver um *rollback*, poderá ocasionar uma cascata de *rollbacks* e consequentemente uma variação grande na utilização. Por isso, Som e Sargent (2000) consideram que a *utilização efetiva* não é adequada quando processos mais rápidos e processos mais lentos não pertencem ao mesmo SG. Isso devido ao fato da utilização ser calculada baseada no número de *rollbacks*, que podem não ocorrer ou podem ocorrer em demasia.

A abordagem de balanceamento baseado em *utilização efetiva* pode ser modificada para equilibrar o progresso de todos os processos em um modelo baseado em

SGs. No trabalho de Reiher e Jefferson (1990), os valores da *utilização efetiva* são utilizados para alterar o mapeamento. Já no trabalho de Som e Sargent (2000), as informações sobre o mapeamento são utilizadas para alterar os valores da *utilização efetiva*. O objetivo é encontrar valores ideais de utilização para igualar a taxa de progresso dos SGs da simulação.

Primeiramente, observa-se a relação entre a *utilização efetiva* e o mapeamento dos processos nos processadores. A utilização observada é usada para fazer o mapeamento dos processos e alcançar o mapeamento desejado. Por outro lado, a mudança no mapeamento dos processos altera o valor de utilização observado.

O mapeamento mostrado na Figura 5 está balanceado. Se objetiva-se reduzir a utilização de L_3, L_1 pode ser movido do Processador 1 para o Processador 2. Suponha que a meta seja reduzir a utilização de L_3 para 0.1 sem mudar a utilização dos demais processos. Com a meta de utilização para L_3 a utilização observada para esse cenário, com apenas L_2 no Processador 1, é: $Processador1 = L_2(0.4)$ e $Processador2 = L_1(0.1), L_3(0.1), L_4(0.2)$. O novo mapeamento também está balanceado, mostrando que o mapeamento pode ser guiado pela meta de utilização.

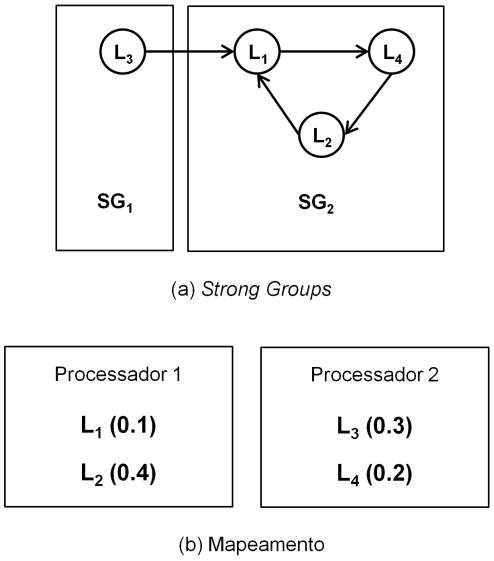

Figura 5: *Strong Groups* e Mapeamento (SOM; SARGENT, 2000).

A questão é como computar a meta de utilização para equilibrar o progresso dos SGs. Esse equilíbrio pode ser conseguido desacelerando os SGs mais rápidos em relação aos SGs mais lentos. Uma vez que a utilização está relacionada com a taxa de progresso, se o objetivo for reduzir a taxa de progresso de um processo em um fator k, deve-se reduzir sua utilização observada em um fator k.

Como a taxa de progresso de todos os processos de um determinado SG se mantém equilibrada, para reduzir a taxa de progresso de um SG em um fator k, a utilização de todos os processos do SG deve ser reduzida em um fator k. Vale ressaltar que, quanto mais rápido um SG for, em relação aos demais SGs, maior será seu fator k e sua meta de utilização será nitidamente menor que a utilização observada. Utilizar os valores de metas de utilização leva a um equilíbrio entre os SGs, onde os mais rápidos são desacelerados e os mais lentos são acelerados.

Não é necessário atingir os valores da meta de utilização após um novo mapeamento. Se os valores de utilização observados após o mapeamento estão próximos aos valores da meta, as taxas de progresso de todos os SGs estarão aproximadamente equilibradas e, consequentemente, a carga também. Se as taxas de progresso ainda estiverem desequilibradas, mais uma interação computando as metas de utilização e remapeamento pode ser feita.

3.5.5 BGE - *Background Execution*

Carothers e Fujimoto (2000) consideraram que a maioria dos estudos anteriores não levaram em consideração qualquer interferência externa ao modelo da simulação, considerando o ambiente totalmente dedicado à simulação. Citam como possíveis interferências os processos dos usuários, a adição ou remoção de novos processadores ao ambiente, as características de desempenho dos diferentes processadores e até mesmo o próprio modelo de simulação, onde, em alguns casos, o tempo de CPU gasto para processar um evento pode variar entre os LPs e a população de eventos dos processos pode ser diferente. Acreditam que uma simulação distribuída bem balanceada, em um ambiente totalmente dedicado, pode tornar-se

mal balanceada em um ambiente não dedicado e que ambientes não dedicados, como as redes de computadores, continuará sendo um paradigma comum.

Os autores observam que os problemas de desempenho, mesmo que originados de diferentes fontes, são similares na maneira em que afetam a simulação. Definem que, para uma simulação otimista estar equilibrada, o tempo real necessário para avançar um processo em uma unidade de tempo lógico deve ser o mesmo para todos os processadores da simulação.

Executar uma simulação usando o protocolo *Time Warp* em uma plataforma NOW estende os problemas de balanceamento de três maneiras. Primeiro, as cargas impostas ao sistema por computações externas não podem ser controladas pelo gerenciador de carga da simulação. Segundo, o conjunto dos processadores utilizados na simulação pode mudar de maneira imprevisível e, por último, os processadores do conjunto de processadores utilizados podem ter características de desempenho diferentes.

A diferença fundamental entre os outros mecanismos e o BGE é a não utilização do tempo lógico para o balanceamento de carga. A política de balanceamento BGE trabalha com duas componentes principais:

- **Alocação dinâmica de processadores**: que determina o conjunto de processadores que fará parte da simulação, sendo que esse conjunto pode variar dinamicamente durante a simulação. Essa componente cuida da adição e/ou remoção de processadores ao ambiente da simulação;
- **Balanceamento de carga**: que gerencia a carga e migra processos entre os processadores considerando a variação da carga externa nos processadores da simulação.

Devido ao fato da migração de processos individualmente ser, computacionalmente, mais caro do que migrar um conjunto de processos, o BGE trabalha com a migração de conjuntos de processos. Sendo assim, agrupa os processos em *clusters*

e trabalha cada conjunto de processos como sendo uma unidade atômica, tanto para cálculo de carga quanto para migração. Para diminuir a quantidade de migrações, procura manter processos que se comunicam mais no mesmo processador, agrupando-os no mesmo *cluster*. Assumem conhecer bem o modelo da simulação para que sejam criados os *clusters*, já que, uma vez em um *cluster*, o processo permanece nele até o fim da simulação.

A métrica central do algoritmo é a *Processor Advance Time* (PAT), que é o montante de tempo real necessário para um processador avançar uma unidade do tempo lógico, não considerando eventos que sofreram *rollback*. Considera apenas o trabalho efetivo, que não será mais desfeito. A razão para não considerar a computação desfeita é devido ao fato de que ela é dependente da aplicação e nem sempre é causado pela carga não estar balanceada. Segundo os autores, considerar a computação desfeita para analisar o balanceamento de carga pode gerar decisões erradas.

Quanto maior o valor de PAT, mais sobrecarregado está o processador. A política move *clusters* de processadores com valores de PAT altos para processadores com PAT menores, a fim de minimizar a diferença dos PATs entre os processos da simulação. O *cluster* a ser migrado é escolhido com base na afinidade (troca de mensagens) em relação ao processador que receberá o *cluster*. A migração só é realizada se a diferença entre os PATs de quaisquer pares de processadores excederem uma porcentagem predefinida. Essa porcentagem é definida pelo usuário da simulação e evita a realização de migrações que iriam causar pouco ou nenhum benefício à simulação.

Outra métrica, a *Cluster Advance Time* (CAT), é utilizada para estimar o tempo real que um *cluster* gasta para avançar uma unidade do tempo lógico. A métrica CAT é calculada de acordo com a Equação 3.9.

$$CAT_{c,i} = \frac{CPU_{c,i}}{\Delta GVT} , \tag{3.9}$$

onde $CPU_{c,i}$ é o montante de tempo de CPU gasto pelo *cluster* c, alocado no

processador i, para processar eventos salvos no último intervalo de tempo entre os cálculos de carga (*TSchedule*) e ΔGVT é o incremento do GVT no último *TSchedule*.

A métrica PAT é o somatório das métricas CAT dos *cluster* alocados em um determinado processador e é calculada de acordo com a Equação 3.10.

$$PAT_i = \sum_{i=0}^{C_i - 1} \frac{CAT_{c,i}}{TWFrac_i} \ , \tag{3.10}$$

onde $TWFrac_i$ é a fração do tempo total de CPU gasto pela simulação no processador i no último *TSchedule*, Equação 3.11.

$$TWFrac_i = \frac{TotalCPU_i}{TSchedule} \ , \tag{3.11}$$

sendo $TotalCPU_i$ o tempo total de CPU dado pelo processador no último *TSchedule*.

A componente que cuida do gerenciamento dos processadores retira do conjunto de processadores utilizáveis pela simulação aqueles processadores que perderam todos os seus processos devido às migrações realizadas. Um processador só retorna ao conjunto se sua carga se tornar duas vezes menor do que quando foi retirado do conjunto.

Com exceção do desbalanceamento causado por problemas de comunicação, o BGE se propõe a combater todas as outras causas de desbalanceamento em uma simulação distribuída executando sobre o protocolo *Time Warp* apontadas pelos autores.

Segundo os testes executados, o algoritmo proposto é capaz de:

- Identificar novos processadores disponíveis e adicioná-los, em tempo de execução, à simulação, migrando partes da simulação para esses processadores;
- Identificar processadores muito carregados por agentes externos e retirá-los da simulação, redistribuindo a carga desses processadores aos processadores

remanescentes;

- Equilibrar a carga nos processadores à medida que há variação na carga de trabalho. Seja essa variação de carga inerente à simulação ou externa.

Os resultados apresentados pelos autores, para diferentes modelos de simulação e com variação interna na carga, variam de 29% a 127% de melhoria no *speedup*. Para variação externa na carga os resultados apresentam melhorias que vão de 24% a 260%. Os resultados apresentados quando o ambiente era composto por processadores com diferentes capacidades de processamento variam de 27,5% a 70%. A última análise apresentada foi realizada sobre simulações executando em um ambiente misto, onde houve variação de carga interna e processadores com capacidades diferentes. Os resultados para essa análise final mostram melhorias de 100%. As comparações foram realizadas executando simulações com e sem o uso do algoritmo de carga BGE.

3.5.6 Outros Algoritmos

Low (2002) apresentou um algoritmo que realiza o balanceamento da carga de processamento, da carga de comunicação e do *lookahead* dos processadores de uma simulação distribuída. O algoritmo de balanceamento é executado a cada n ($n >= 1$) cálculos de GVT. As métricas utilizadas para análise de carga no ambiente são o *lookahead* entre os processos (o quanto um processo está a frente de um processo comunicante), a carga de processamento e o número de mensagens trocadas entre os processos.

Jiang, Anane e Theodoropoulos (2004) criaram uma variação do algoritmo BGE, considerando a latência de comunicação e o padrão de comunicação entre os processos. Definem duas métricas adicionais, a *Processor-to-Processor Communication* (PPC) e a *Cluster-to-Processor Communication* (CPC). As métricas PPC e CPC indicam, respectivamente, o total de mensagens trocadas entre dois processadores (entre quaisquer dois processos alocados nesses dois processadores) e o

total de mensagens trocadas entre um *cluster* e um processador (qualquer processo do processador em questão). Baseando-se nesses valores, o algoritmo migra carga de um processador P_i com alta PAT para um processador P_j com baixa PAT de acordo com a Equação 3.12.

$$PCS_{ij} = \frac{PPC_{ij} \cdot LATENCY_{ij}}{PAT_j} \tag{3.12}$$

onde PCS_{ij} é a velocidade de comunicação de processador (*Processor Communication Speed*) do processador P_j em relação à conexão (*link*) de P_i a P_j. A migração ocorrerá do processador P_i para o processador P_j de maior PCS_{ij}, ou seja, a carga será migrada para um processador menos carregado (com um valor baixo de PAT) que tenha um nível alto de comunicação com o processador de origem P_i. O *cluster* que migrará de P_i para P_j é aquele de maior $(CAT \cdot CPC)$, ou seja, aquele com alta carga e que contribui mais para o tráfico de mensagens entre os processadores P_i e P_j.

Outros algoritmos que consideram a carga da computação e o padrão de comunicação podem ser vistos em (PESCHLOW; HONECKER; MARTINI, 2007), (AJALTOUNI; BOUKERCHE; ZHANG, 2008) e (DE GRANDE; BOUKERCHE, 2011).

Meraji, Zhang e Tropper (2010) apresentaram um algoritmo de balanceamento de carga dinâmico para simulação otimista baseado no aprendizado de máquina. Nessa abordagem, um agente interage com o ambiente. A cada interação com o ambiente, o agente determina o estado (s) do ambiente e determina uma ação (a) a ser executada, então o ambiente retorna um sinal (r) indicando a eficácia da ação executada. Como vantagens dessa abordagem, pode ser destacado a não necessidade de conhecimento do ambiente, pois a política é baseada em *feedbacks*. Como desvantagem, o custo de implementação e o *overhead*.

O algoritmo proposto por Meraji, Zhang e Tropper (2010) trabalha com a junção de dois algoritmos, um que visa equilibrar a carga entre os processadores, **algoritmo de carga**, e o outro que visa minimizar a carga de comunicação, **algoritmo de comunicação**. Esses dois algoritmos fazem uso de quatro parâmetros,

que são computados durante os intervalos entre duas execuções do algoritmo de balanceamento:

1. **Carga do processo (*LpLoad*)**: a carga de cada processo é determinada como o número de eventos processados;

2. **Carga do processador (*PLoad*)**: é a soma das *LpLoads* dos processos alocados no processador;

3. **Carga de comunicação do processo (*LpComm*[])**: é representada por um vetor de tamanho $n-1$, onde n representa o número de processadores na simulação. Cada elemento do vetor é o número de mensagens que o processo enviou para os outros processadores da simulação;

4. **Carga de comunicação do processador (*PComm*[])**: também é representada por um vetor de tamanho $n-1$, cada elemento do vetor é o número de mensagens que o processador, soma das mensagens enviadas por todos os processos alocados no processador, enviou para os outros processadores da simulação.

No algoritmo de carga, cada processador envia seus valores dos parâmetros *PLoad* e *PComm*[] para um processo mestre, o processo mestre determina os P processadores mais e menos carregados (maiores e menores *PLoads*), sendo P um parâmetro configurado pelo usuário. Para cada par de processadores, mais e menos carregados, o processador mais carregado seleciona L, outro parâmetro fornecido pelo usuário, processos e os envia para seu correspondente processador menos carregado. Os processos selecionados para migrarem são aqueles que possuem os maiores valores de *PComm*[] em relação ao processador de destino da migração. No algoritmo de comunicação, cada processador P_i envia os valores dos parâmetros $PLoad_i$ e $PComm_i$[] para o processo mestre. O processo mestre determina o maior valor $PComm_i[j]$ em $PComm_i$[]. Para os processos P_i e P_j que tiveram a maior comunicação entre as últimas duas execuções do algoritmo de balanceamento, de

acordo com os valores de $PComm_i[j]$ e $PComm_j[i]$, o algoritmo migra processos entre esses dois processadores, de forma que o processador de maior *PLoad* é o processador que envia processos. Os processos selecionados para migrarem são aqueles que também possuem os maiores valores de $PComm[]$ em relação ao processador de destino da migração.

Um último parâmetro utilizado pelo algoritmo determina qual dos dois algoritmos será utilizado, o algoritmo de carga ou o algoritmo de comunicação. Para o algoritmo de aprendizado, há quatro estados possíveis na simulação:

1. ***BcompBcomm***: carga e comunicação balanceadas;
2. ***BcompUcomm***: carga balanceada e comunicação desbalanceada;
3. ***UcompBcomm***: carga desbalanceada e comunicação balanceada;
4. ***UcompUcomm***: carga e comunicação desbalanceadas.

No primeiro estado, a computação está balanceada. No segundo estado, é executado o algoritmo de comunicação. No terceiro estado é executado o algoritmo de carga e no quarto estado, onde ambos os parâmetros estão desbalanceados, é escolhida a melhor ação a ser tomada de acordo com os parâmetros, que pode ser executar o algoritmo de comunicação ou o algoritmo de carga. Mais detalhes podem ser vistos em (MERAJI; ZHANG; TROPPER, 2010).

Outros trabalhos, focados em balancear a carga de simulações distribuídas em ambientes *multi-core* de memória compartilhada, podem ser vistos em (CHEN et al., 2011) e (CHILD; WILSEY, 2012).

3.6 Considerações Finais

Uma importante revisão sobre escalonamento de processos foi realizada nesse capítulo. Começando pela definição do que é um processo e pela diferenciação entre

os níveis de escalonamento, passando ainda por conceitos relacionados a índices de carga e como cada ambiente possui particularidades em relação à sua carga.

Um estudo sobre a classificação dos algoritmos de escalonamento, aqueles voltados às simulações que executam sobre protocolos otimistas, também foi realizada. Esse estudo justifica a escolha dos algoritmos de escalonamento apresentados, considerando adequados aqueles que trabalham sob uma abordagem dinâmica ou mista.

O capítulo ainda apresentou conceitos básicos sobre a migração de processos, mecanismo necessário quando se trata de abordagens dinâmicas para o escalonamento de carga.

Por fim, o capítulo apresentou os principais fatores que causam o desbalanceamento de carga em ambientes de Simulação Distribuída. O entendimento desses fatores é importante para a compreensão dos algoritmos de escalonamento para Simulação Distribuída otimista, também apresentados nesse capítulo e dos quais foram extraídas as métricas avaliadas.

O próximo capítulo apresenta as adaptações realizadas na implementação do protocolo *Time Warp* e no mecanismo de migração de processos para atender algumas necessidades do mecanismo de balanceamento de carga proposto.

4 Adaptações na implementação do protocolo *Time Warp* e migração de processos

Junqueira (2012) apresentou dois mecanismos para migração de processos em aplicações de simulação distribuída baseados no protocolo *Time Warp*. Um mecanismo de migração coletiva onde, durante uma migração, ocorre o encerramento e recriação de todos os processos da simulação e outro mecanismo de migração individual onde, durante uma migração, apenas os processos que realmente irão migrar são encerrados e recriados posteriormente. Junqueira (2012) demonstrou, teoricamente, que os dois mecanismos propostos podem melhorar o desempenho de uma simulação.

Para implementar os mecanismos de migração, Junqueira (2012) fez uso da implementação inicial do protocolo *Time Warp* desenvolvida por Azevedo (2012), que realizou um trabalho de projeto e implementação dos protocolos de sincronização para simulações distribuídas, *Time Warp* e *Rollback* Solidário. Azevedo (2012) também especificou e desenvolveu um *framework* para o desenvolvimento de aplicações de simulação e realizou um estudo comparativo entre os dois protocolos.

Apesar de Azevedo (2012) ter desenvolvido a versão final de seu projeto com a plataforma Java, as primeiras implementações do protocolo *Time Warp* foram desenvolvidas com a linguagem de programação C/C++. Por questões de paralelismo nos desenvolvimentos dos projetos de Azevedo (2012) e Junqueira (2012),

este realizou seu projeto de implementação sobre a primeira implementação do protocolo *Time Warp* desenvolvida por Azevedo (2012), essa desenvolvida em C/C++ e com o ambiente de passagem de mensagem MPI, com a implementação *OpenMPI* na versão 1.5.4.

Para aproveitar os mecanismos de migração de processos desenvolvidos por Junqueira (2012), e assim agilizar o desenvolvimento do presente trabalho, toda nova implementação, melhoria e/ou adaptação desenvolvidas foram realizadas sobre o código do projeto de Junqueira (2012). Devido a alguns requisitos do projeto, principalmente em relação aos dados que são necessários para o cálculo das métricas que guiam o algoritmo de escalonamento, foi observada a necessidade de algumas adaptações nas implementações de Junqueira (2012) e Azevedo (2012). As adaptações realizadas são apresentadas nas próximas seções.

4.1 Arquitetura original do ambiente

Em Junqueira (2012), o ambiente de execução de simulações distribuídas é controlado por um processo mestre, responsável por iniciar o ambiente, criar os processos da simulação, recolher dados sobre a simulação, executar os mecanismos de balanceamento de carga, migrar os processos quando necessário e finalizar a simulação.

Devido à estabilidade do código de migração coletiva desenvolvido por Junqueira (2012), quando o presente trabalho foi iniciado, e também devido ao fato do mecanismo de migração não interferir nas análises comparativas de desempenho das métricas de balanceamento de carga usadas pelo algoritmo de escalonamento, o mecanismo de migração usado no projeto foi o mecanismo de migração coletiva.

Segundo Junqueira (2012), as principais atividades do processo mestre na migração coletiva de processos são:

1. **Aplicar o algoritmo de balanceamento**: quando o processo mestre é

inicializado, a simulação ainda não iniciou, havendo um mapeamento estático dos processos da simulação;

2. **Iniciar ou reiniciar os processos**: de acordo com o resultado do algoritmo de balanceamento, o processo mestre criará ou reiniciará os processos nos processadores adequados;

3. **Iniciar um temporizador**: durante este tempo, o processo mestre permanece ocioso deixando a simulação executar sem interferência;

4. **Iniciar a coleta das métricas de desempenho**: o processo mestre solicita aos processos da simulação que coletem as informações necessárias, que depende do algoritmo de balanceamento empregado;

5. **Receber as informações sobre a simulação**: após solicitar informações dos processos sobre a simulação, o mestre aguarda até receber os dados de todos os processos. Com as informações, o mestre pode ainda decidir se alguma condição de término foi satisfeita e, em caso positivo, todos os processos da simulação são informados de que devem encerrar sua execução;

6. **Aplicar o algoritmo de balanceamento**: caso nenhuma condição de parada tenha sido satisfeita, o algoritmo de balanceamento é executado. O algoritmo deve informar a necessidade de migração e determinar onde cada processo deve ser reiniciado;

7. **Anunciar migração**: havendo necessidade de migração, todos os processos são informados que sofrerão migração;

8. **Aguardar o encerramento dos processos**: em caso de migração, os processos da simulação deverão ser encerrados e reiniciados. Para manter a consistência das informações que ainda possam estar sendo trocadas entre os processos, o processo mestre aguarda o término de todos os processos da simulação;

9. **Retornar ao passo dois**.

O diagrama de atividades da Figura 6 ilustra as principais ações do processo mestre.

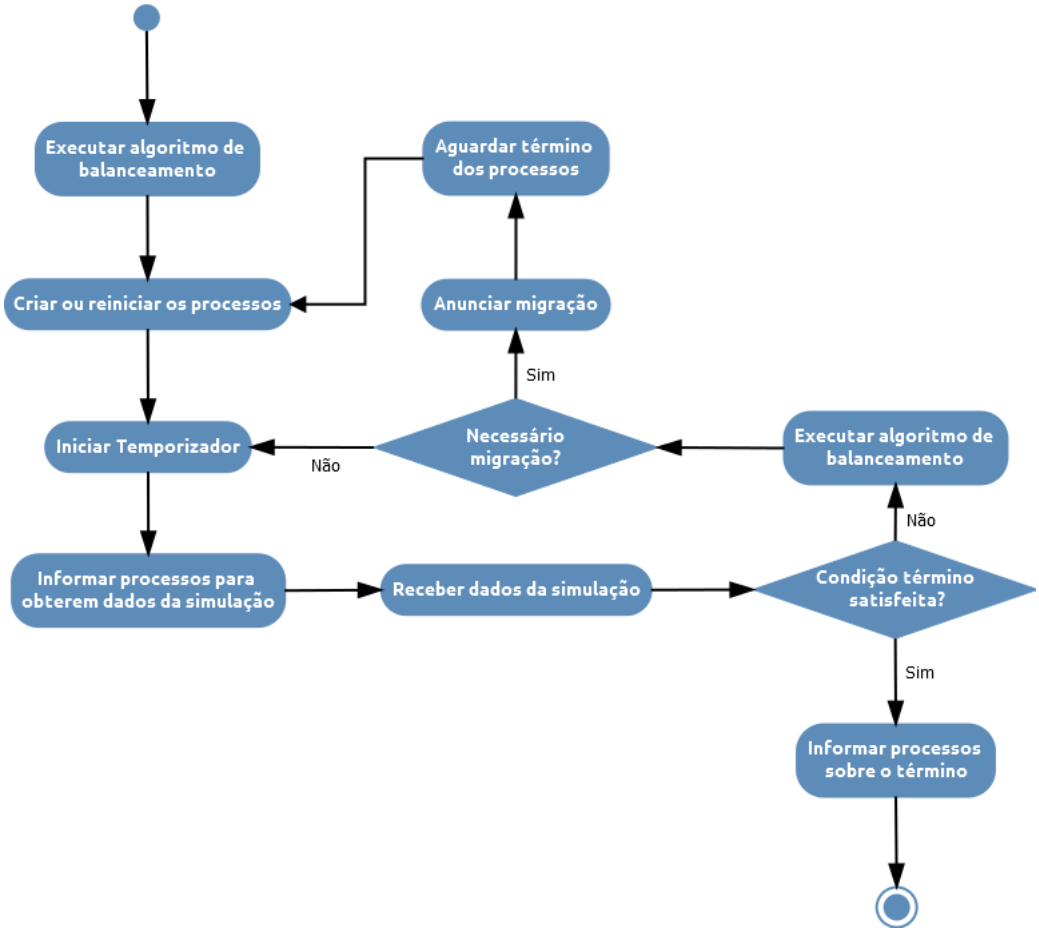

Figura 6: Diagrama de atividades do processo mestre na migração coletiva de processos (JUNQUEIRA, 2012).

Devido a adição do algoritmo do cálculo do GVT, uma atividade foi adicionada entre as atividades 3 (iniciar temporizador) e 4 (iniciar coleta das métricas de desempenho). Os detalhes dessa atividade são mostrados na Seção 4.2.

Os processos da simulação possuem uma *thread* principal, responsável por executar a simulação, e uma *thread* de sincronismo, responsável pelo sincronismo com

o processo mestre, pelo envio das métricas necessárias ao algoritmo de balanceamento e pela interação com a *thread* principal.

Segundo Junqueira (2012), as principais atividades da *thread* principal na migração coletiva de processos são:

1. **Inicializar a thread de sincronismo**: antes de entrar no *loop* principal da simulação, a *thread* de sincronismo é inicializada;
2. **Verificar se é a primeira execução do processo**: caso o processo já tenha passado por migrações, suas variáveis e mensagens recebidas durante o processo de migração devem ser recuperadas. Essas informações são salvas em arquivos previamente criados e recuperadas no momento da recriação dos processos. Esses arquivos são salvos em um servidor de arquivos e independente do *host* que cada processo seja reiniciado, terão acesso as suas informações;
3. **Verificar condição de término**: uma variável fica responsável por informar se o critério de parada da simulação foi satisfeito. A *thread* de sincronismo atualiza essa variável de acordo com informações que recebe do processo mestre;
4. **Verificar se há necessidade de migração**: outra variável fica responsável por informar se o processo será migrado. A *thread* de sincronismo também atualiza essa informação que é enviada pelo processo mestre;
5. **Realizar a iteração**: se a condição de término não foi satisfeita e não houver necessidade de migração, a próxima iteração da simulação é executada;
6. **Retorna ao passo 3**.

O diagrama de atividades da Figura 7 ilustra as principais atividades da *thread* principal dos processos da simulação.

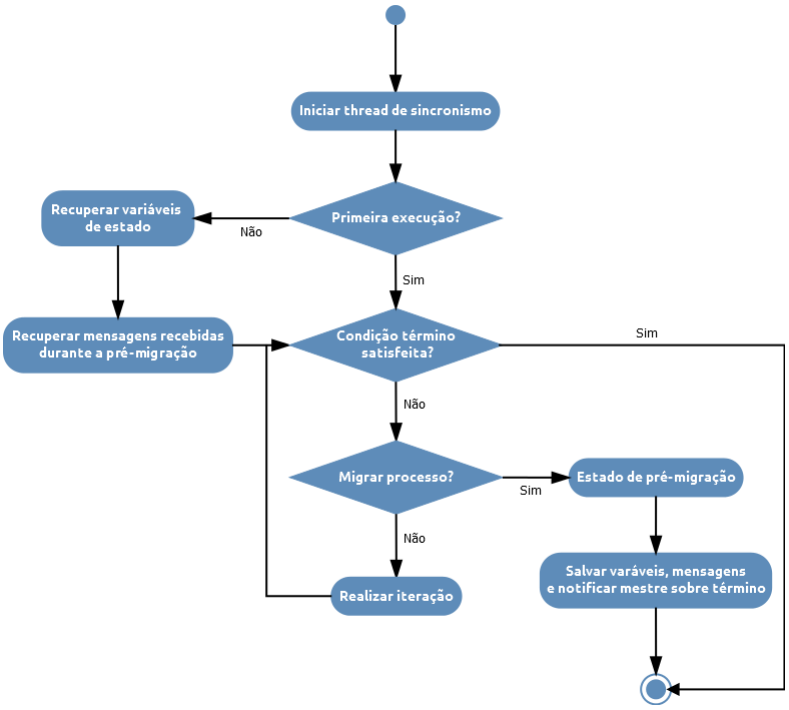

Figura 7: Atividades da *thread* principal dos processos da simulação (JUNQUEIRA, 2012).

O estado de pré-migração, citado no diagrama da Figura 7, foi introduzido por Junqueira (2012) e é um estado entre: (1) a notificação enviada pelo processo mestre aos processos da simulação indicando que haverá uma migração e (2) o término dos processos da simulação propriamente dito. Quando são informados de que serão migrados, os processos não podem encerrar imediatamente, pois isso levaria a perda das mensagens da simulação em trânsito na rede. No estado de pré-migração, os processos apenas recebem as mensagens em trânsito, não enviam mais mensagens da simulação aos outros processos, apenas enviam uma mensagem a cada processo da simulação informando que irão migrar. Cada processo aguarda receber de todos os outros processos a mensagem de que estão migrando, só assim o processo pode terminar, tendo a certeza de que não há nenhuma mensagem pendente na rede endereçada a ele. O processo de pré-migração pode ser visto com mais detalhes em (JUNQUEIRA, 2012).

A *thread* de sincronismo presente nos processos da simulação tem a função de receber as solicitações do processo mestre e aplicá-las sobre o processo da simulação. Junqueira (2012) lista as seguintes atividades para a *thread* de sincronismo:

1. **Verificar se há mensagens para o processo**: se não houver mensagens, a *thread* dorme por um pequeno período de tempo;
2. **Identificar instrução recebida**: a *thread* identifica o tipo de mensagem recebida do processo mestre. Se for uma mensagem informando que haverá migração, a *thread* modifica a variável que identifica essa situação e como a *thread* principal tem acesso a essa variável, dá inicio ao processo de migração. O mesmo ocorre caso a mensagem recebida seja de término da simulação, onde é iniciado o processo de encerramento da simulação.
3. **Obter dados da simulação**: se a mensagem recebida for uma solicitação de informações da simulação, a *thread* coleta as informações da simulação, geralmente dados necessários para o algoritmo de balanceamento, e envia para o processo mestre;
4. **Retorna ao passo 1**.

O diagrama de atividades da Figura 8 ilustra as principais atividades da *thread* de sincronismo dos processos da simulação.

4.2 Implementação do algoritmo de Mattern para cálculo do GVT

Algumas das principais métricas utilizadas pelos algoritmos de balanceamento de carga para simulação distribuída são calculadas utilizando o GVT da simulação. Dessa forma, como ainda não havia sido implementado, o cálculo do GVT precisou ser adicionado ao projeto.

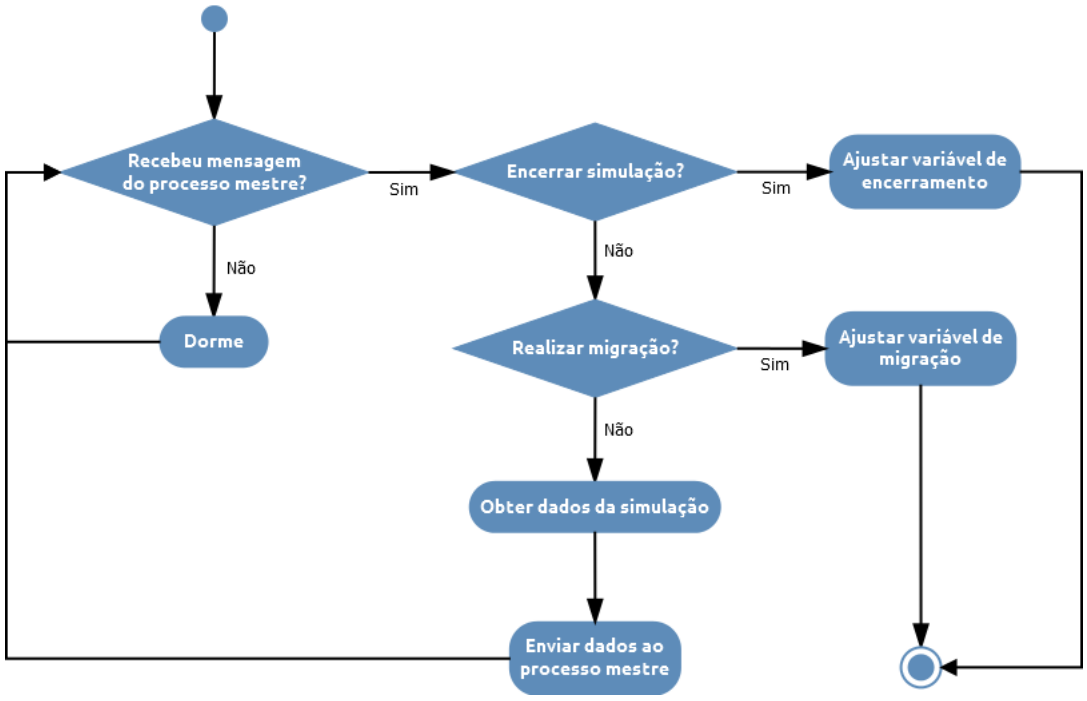

Figura 8: Atividades da *thread* de sincronismo dos processos da simulação (JUNQUEIRA, 2012).

Devido aos fatos de não interferir no otimismo da simulação e de não precisar de mensagens de confirmação, o algoritmo de Mattern (1993), descrito na Seção 2.5.3.2, foi o algoritmo de cálculo de GVT utilizado.

O cálculo do GVT é coordenado pelo processo mestre que, através da troca de mensagens, solicita e envia informações aos processos da simulação. Algumas das informações enviadas e solicitadas pelo processo mestre estão relacionadas com o cálculo do GVT.

Inicialmente, todos os processos são marcados com a cor branca. As mensagens enviadas carregam a cor do processo, a cor é usada para determinar a fase em que o processo está e se todas as mensagens brancas enviadas foram recebidas. A interação entre o processo mestre e os processos da simulação para a coleta das informações necessárias para calcular o GVT ocorre da seguinte maneira:

1. **Envio de mensagem de corte branco**: o processo mestre envia uma

mensagem de corte branco a todos os processos da simulação, solicitando que informem a quantidade de mensagens brancas enviadas a cada processo da simulação;

2. **Recebimento de mensagem de corte branco**: ao receber a mensagem de corte branco, cada processo se coloca na cor vermelha. Como resposta, envia ao processo mestre um vetor de inteiros, uma entrada para cada outro processo da simulação, indicando quantas mensagens brancas foram enviadas antes do corte (mudança de cores). A partir desse momento, o processo começa a armazenar o menor *timestamp* entre as mensagens vermelhas enviadas;

3. **Resposta de mensagem de corte branco**: após receber a resposta de todos os processos, o processo mestre sabe que todos os processos da simulação estão na cor vermelha. Nesse momento, utilizando das informações de mensagens brancas enviadas que recebeu na resposta, o processo mestre calcula o total de mensagens brancas que foram enviadas a cada processo;

4. **Envio do total de mensagens brancas enviadas**: após calcular quantas mensagens brancas foram enviadas a cada processo da simulação, o processo mestre envia essa informação a todos os processos;

5. **Recebimento do total de mensagens brancas enviadas**: ao receber a informação de quantas mensagens brancas foram enviadas a ele, cada processo da simulação verifica se já recebeu a quantidade de mensagens brancas informada. Caso já tenha recebido, o processo envia ao processo mestre o menor valor entre: (1) seu LVT e (2) o menor *timestamp* entre as mensagens vermelhas enviadas. Caso o processo ainda não tenha recebido todas as mensagens brancas enviadas, ele deve aguardar até que receba;

6. **Resposta do total de mensagens brancas enviadas**: após receber a resposta de todos os processos, o processo mestre tem a informação do menor valor estimado para o LVT de cada processo. Esse valor pode ser o próprio

LVT do processo ou o *timestamp* de alguma mensagem vermelha enviada, desde que seja menor que o LVT do processo. Com essas informações, o processo mestre calcula o GVT da simulação, que é o menor entre esses valores recebidos dos processos;

7. **Envio de mensagem de corte vermelho**: o processo mestre envia uma mensagem de corte vermelho a todos os processos da simulação. Junto a essa solicitação, o processo mestre também envia o GVT da simulação recém calculado;

8. **Recebimento de mensagem de corte vermelho**: ao receber a mensagem de corte vermelho, cada processo se coloca na cor branca. Como resposta, cada processo envia ao processo mestre as informações relativas às métricas de balanceamento de carga. É nesse momento que as informações sobre as métricas são enviadas ao mestre. No entanto, essas informações são calculadas a todo instante pelos processos da simulação. A forma como cada métrica é calculada será detalhada na Seção 5.2;

9. **Resposta de mensagem de corte vermelho**: após receber a resposta de todos os processos e verificar que nenhuma condição de parada foi satisfeita, o processo mestre usa as informações da métrica selecionada para chamar o algoritmo de balanceamento.

10. **Processo mestre aguarda um período de tempo e retorna ao passo 1**.

A interação entre o processo mestre e os processos da simulação para coletar informações e calcular o GVT é ilustrada na Figura 9.

4.3 Gerenciamento de memória

Para não limitar a estrutura dos modelos, não limitar o tempo de execução das simulações e principalmente não prejudicar o desempenho da simulação, foi

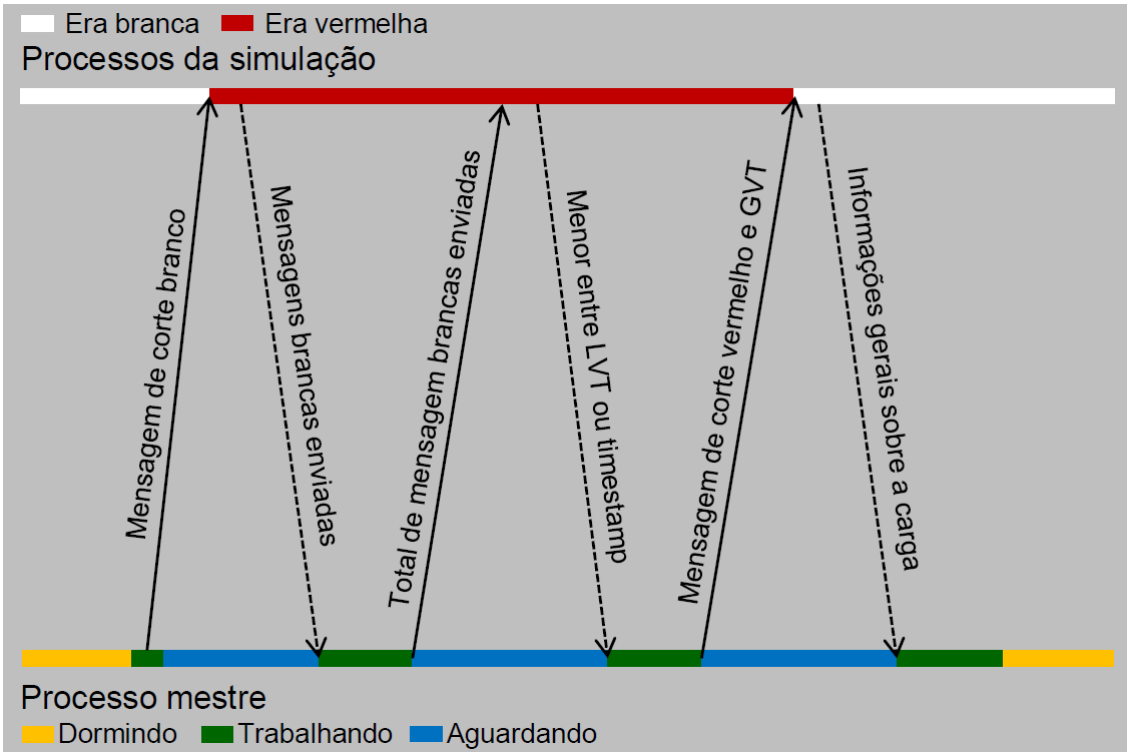

Figura 9: Interação entre o processo mestre e os processos da simulação para coletar informações.

implementado um gerenciamento de memória no ambiente.

O gerenciamento de memória consiste basicamente no controle da lista de estados globais salvos, necessários em caso de *rollbacks*, e da lista de mensagens enviadas, necessárias para envio de anti-mensagens.

Os estados armazenados na lista de estados globais salvos e as mensagens enviadas armazenadas na lista de mensagens enviadas, cujos LVTs são menores que o GVT, nunca seriam utilizados em um possível *rollback*. Portanto, esses estados e mensagens são retirados das suas respectivas listas e a memória alocada é liberada.

O algoritmo de controle da memória é executado toda vez que um novo GVT é calculado, mantendo o gasto de memória controlado durante a simulação.

4.4 Salvamento das informações da simulação

Todas as informações necessárias para restaurar o estado dos processos após uma migração já eram salvos pelo ambiente implementado por Azevedo (2012). Algumas outras informações, como, por exemplo, o total de eventos processados, o total de mensagens enviadas, o total de mensagens recebidas etc, também eram coletadas pelo ambiente. Porém, essas informações eram zeradas a cada migração e não eram mantidas em memória ou em disco.

Para possibilitar as análises sobre o desempenho do algoritmo de balanceamento de carga, foi necessário realizar o salvamento dessas informações da simulação antes de cada migração. Isso possibilitou ter, no final da simulação, um resumo geral da mesma, que foi utilizado para a avaliação do algoritmo de balanceamento de carga.

Na prática, essas informações não são persistidas, elas são apenas mantidas na memória do processo mestre, que mantém as informações de execução de cada processo da simulação. Antes de migrar, um processo da simulação envia suas informações para o processo mestre, que soma essas informações com as informações anteriormente recebidas, mantendo assim as informações de cada processo e, consequentemente, da simulação como um todo.

As informações de cada processo da simulação, mantidas pelo processo mestre, e utilizadas para análise do desempenho da simulação são as seguintes:

1. **O total de eventos processados**: o total de eventos processados por cada processo é utilizado para o cálculo de algumas métricas de balanceamento de carga. Também é utilizado para ter a informação do total de eventos processados em toda a simulação. Essa informação é importante para o cálculo da eficiência do algoritmo de balanceamento;

2. **O total de eventos cancelados**: assim como o total de eventos processados, o total de eventos cancelados é utilizado para o cálculo de algumas

métricas de balanceamento de carga, do total de eventos cancelados em toda a simulação e da eficiência do algoritmo de balanceamento;

3. **Total de mensagens enviadas**: utilizado para possíveis análises no modelo da simulação;
4. **Total de mensagens recebidas**: utilizado para possíveis análises no modelo da simulação;
5. **Total de anti-mensagens enviadas**: utilizado para possíveis análises no modelo da simulação;
6. **Total de anti-mensagens recebidas**: utilizado para possíveis análises no modelo da simulação;
7. **Total de *rollbacks***: utilizado para análise de desempenho do algoritmo de balanceamento de carga;
8. **Total de mensagens *straggler***: utilizado para possíveis análises no modelo da simulação e para o cálculo da métrica para balanceamento de carga proposta neste trabalho.

Através dessas informações, foi possível avaliar o desempenho das simulações executadas.

4.5 Considerações Finais

Esse capítulo apresentou o ambiente sobre o qual as simulações realizadas foram executadas, mostrando em detalhes a estrutura do ambiente, seus componentes e a forma de interação entre eles.

Apresentou também as adaptações importantes que foram necessárias. Adaptações relacionadas à coleta de informações para o cálculo dos índices de carga,

coleta de informações para avaliar o desempenho da simulação e para o gerenciamento da memória, fator importante por impactar diretamente no desempenho do mecanismo de migração de processos.

O próximo capítulo apresenta o algoritmo de balanceamento proposto para avaliação das métricas e também as métricas que foram avaliadas.

5 Algoritmo de balanceamento e métricas avaliadas

As métricas avaliadas foram extraídas de algoritmos clássicos de balanceamento de carga para simulação distribuída otimista que utilizam o protocolo *Time Warp*.

Esse trabalho também propôs uma métrica para ser usada em algoritmos de balanceamento de carga para simulação distribuída. A métrica foi chamada de *Expected Effective Work* (EEW) e a forma como é calculada será descrita na Seção 5.1.4.

Foi desenvolvido um algoritmo de balanceamento para avaliação das métricas. O algoritmo possui características de alguns dos algoritmos clássicos e trata algumas limitações ao dinamismo de alguns desses algoritmos.

5.1 Métricas avaliadas

As próximas seções listam as métricas que foram avaliadas e descrevem a métrica proposta neste trabalho.

5.1.1 Métrica do relógio local

O algoritmo da métrica do relógio local, descrito na Seção 3.5.2, usa o LVT dos processos como métrica para o algoritmo de balanceamento. Esse algoritmo é citado em diversos trabalhos relacionados e também usado em algumas comparações

de desempenho.

O algoritmo da taxa de avanço da simulação também utiliza a taxa de avanço do relógio local para derivar a fatia de tempo de processamento de cada processo, visando minimizar a diferença entre os LVTs dos processos da simulação.

Um ponto favorável dessa métrica é o fato de nenhum esforço extra ser necessário para seu cálculo, uma vez que é uma informação já presente em um ambiente de simulação, o LVT.

5.1.2 Utilização efetiva do processador

A métrica da *utilização efetiva*, proposta no algoritmo da utilização efetiva do processador, Seção 3.5.1, e também utilizada no algoritmo *Strong Groups*, Seção 3.5.4, também é muito conhecida e citada em trabalhos relacionados, além da possibilidade de ser calculada de duas formas. Pode ser calculada a partir de informações já presentes em um ambiente de simulação, o número de eventos processados e o número de eventos efetivados, ou a partir de informações de ciclos de processamento, o número de ciclos de processamento e o número de ciclos de processamento perdidos devido a *rollbacks*.

5.1.3 PAT - *Processor Advance Time*

A métrica PAT, introduzida no algoritmo BGE, Seção 3.5.5, será avaliada por fazer parte de um dos algoritmos mais completos para o balanceamento de carga em simulação distribuída. O fator que mais pesou para a escolha dessa métrica foi o fato do algoritmo BGE ser bastante complexo e utilizar de diversos mecanismos, incluindo até mesmo a possibilidade de um processador deixar de ser utilizado pela simulação, na tentativa de balancear uma simulação distribuída. A questão a ser tratada é se os bons resultados do algoritmo são devido a sua complexidade e amplitude ou se é devido à qualidade da métrica.

5.1.4 EEW - *Expected Effective Work*

A métrica do trabalho efetivo esperado, proposta nesse trabalho, é calculada com base na taxa de *rollbacks* ocorridos.

A taxa de *rollbacks* é calculada pelo fator de mensagens *stragglers* recebidas em relação ao total de mensagens recebidas, de acordo com a Equação 5.1.

$$RollbackFactor_P = \frac{TotalStraggler_P}{TotalMessages_P} \tag{5.1}$$

A taxa de *rollbacks* permite calcular o montante de trabalho que, provavelmente, será desfeito. Esse montante de trabalho que, possivelmente, será desfeito é a porcentagem do montante de trabalho a fazer que sofrerá *rollbacks*

O trabalho efetivo esperado é a diferença entre o montante de trabalho a fazer pelo montante de trabalho que provavelmente será desfeito. É uma relação entre os eventos impostos a um processo e a quantidade de mensagens *stragglers* recebidas por ele. A Equação 5.2 resume o cálculo do trabalho efetivo esperado.

$$ExpectedEffectiveWork_P = 1 - RollbackFactor_P \tag{5.2}$$

A métrica representa a porcentagem de trabalho realizada por um processo que se tornará trabalho efetivo, ou seja, trabalho que não será desfeito. O cálculo da métrica considera apenas os *rollbacks* primários, aqueles realizados devido ao recebimento de mensagens *stragglers*. *Rollbacks* secundários não são considerados devido ao fato de que não existiriam sem a ocorrência de um *rollback* primário e de que sua quantidade fica dependente de uma série de fatores relacionados ao modelo da simulação, que são difíceis de controlar.

Apesar do cálculo da métrica proposta não estar relacionada diretamente ao relógio local, indiretamente, o relógio local dos processos tendem a estar próximos, uma vez que a diferença entre LVTs de processos comunicantes é a causa das mensagens *stragglers*.

Para a métrica, quanto maior o valor do trabalho efetivo esperado, maior é a carga do processo, pois o processo não está sofrendo com *rollbacks* ou, por qualquer outro motivo, não consegue lidar com a carga imposta a ele e, provavelmente, está causando *rollbacks* em outros processos da simulação.

Os algoritmos que utilizarem essa métrica devem operar na tentativa de equilibrar o trabalho efetivo esperado entre os processadores da simulação, migrando processos dos processadores mais carregados para os processadores menos carregados.

Em geral, os algoritmos de balanceamento de carga para ambientes de simulação distribuída que utilizam o protocolo *Time Warp* buscam reduzir a ocorrência de *rollbacks*. A métrica proposta procura, justamente, trabalhar na raiz desse problema, as mensagens *stragglers*, usando o fator de ocorrência desse tipo de mensagens e tentando controlar fatores que podem causá-las, tais como a variação da quantidade de eventos e do tempo de serviço entre os processos.

5.2 Cálculo das métricas

As métricas avaliadas pertencentes a outros algoritmos foram calculadas exatamente como descrito nos trabalhos originais que as introduziram.

A métrica do **relógio local**, Seção 5.1.1, já é calculada pelo próprio protocolo de sincronização, pois o LVT é uma informação necessária para manter o sincronismo entre os processos. O LVT de cada processo fornece informação suficiente para o cálculo da métrica.

A **utilização efetiva** é calculada em duas versões, conforme descrito na Seção 5.1.2. A utilização efetiva calculada a partir do número de eventos efetivados e do número total de eventos processados, informações já disponíveis no ambiente de simulação, e uma estimativa da utilização efetiva calculada a partir do número total de ciclos de processamento gastos para processamento dos eventos e do número

de ciclos de processamento perdidos com *rollbacks*.

As informações de ciclos de processamento precisaram ser implementadas no ambiente de execução das simulações, para que fosse possível o cálculo dessa métrica e também da métrica PAT.

Para computar os ciclos de processamento gastos para execução dos eventos, foi utilizada a função `clock_gettime`. A função `clock_gettime` permite a obtenção do tempo de relógio em alta resolução (LINUXMANPAGES, 2013a). Essa função retorna a informação de tempo em dois campos, segundo e nanosegundos, bilionésima parte de um segundo (10^{-9} segundos), permitindo a obtenção do tempo por meio de diferentes relógios providos pelo sistema, como por exemplo:

1. `CLOCK_REALTIME`: relógio de tempo real do sistema;
2. `CLOCK_PROCESS_CPUTIME_ID`: relógio de alta resolução que a CPU provê para cada processo;
3. `CLOCK_THREAD_CPUTIME_ID`: relógio de alta resolução que a CPU provê para cada *thread*.

Outros relógios podem ser fornecidos pelo sistema, mas esses três são suficientes para obtenção de tempos de execução em nível de sistema, processos e *threads*.

Para obter o tempo de execução de uma função é necessário realizar duas chamadas a função `clock_gettime`. Uma antes da função ser chamada e outra após a execução da função, subtraindo os valores para obter o tempo de execução, que no caso dessa função pode ser na resolução de nanosegundos. Foi necessário a utilização dessa função porque as funções clássicas de *timers* não oferecem a resolução necessária e, na prática, não conseguiam identificar o tempo de execução de blocos de códigos de execução muito rápida.

Fazendo uso dessa função, os ciclos de processamento utilizados pelo processo para execução de todos os eventos foi contabilizado, assim como os ciclos de pro-

cessamento gastos pelo processo na execução de *rollbacks*, fornecendo assim as informações necessárias para o cálculo da estimativa da utilização efetiva.

O cálculo da métrica PAT, Seção 5.1.3, além de utilizar a função de tempo `clock_gettime`, também utiliza a função `getrusage`. A função `getrusage` retorna diversas estatísticas de uso sobre o processo chamador, tais como tempo de processamento e memória utilizada, sendo útil para uma análise sobre os recursos utilizados pelo processo (LINUXMANPAGES, 2013b). Para usar a função, basta chamá-la quando se deseja obter as informações do processo necessárias para alguma análise. Essa função foi utilizada para calcular o tempo total de CPU gasto pelos processos. Como a resolução dessa informação é a nível de segundos, a função foi suficiente. Essas funções são as mesmas que foram utilizadas por Carothers e Fujimoto (2000).

Originalmente, o algoritmo de Carothers e Fujimoto (2000) agrupa os processos em *clusters* para calcular a métrica CAT, como o algoritmo que será apresentado na Seção 5.3 não faz qualquer análise estática do modelo da simulação, a métrica CAT é calculada individualmente para cada processo e os processos também são migrados individualmente.

Seguindo a abordagem de cálculo realizada por Carothers e Fujimoto (2000), a computação efetiva é calculada somando os ciclos de processamento utilizados para inserção e remoção de eventos na fila de eventos futuros, salvamento de estado global e processamento dos eventos. Desse montante de ciclos, são subtraídos os ciclos de processamentos utilizados na execução de *rollbacks*. A métrica CAT é calculada dividindo o total de ciclos de computação efetiva pelo total de ciclos dado ao processo. Finalmente, a métrica PAT de um determinado processador é o somatório das métricas CAT de todos os processos alocados no processador.

Conforme descrito na Seção 5.1.4, a métrica EEW também utiliza dados da simulação para ser calculada.

Conforme será descrito na Seção 6.1, as métricas, assim como os parâmetros

necessários para obtê-las, são calculadas em intervalos de tempo regulares. Esse intervalo de tempo é determinado pelo processo mestre que solicita as informações aos processos da simulação. A cada início de interação do processo mestre com os processos da simulação, os processos da simulação zeram os fatores utilizados para calcular as métricas, ou seja, as métricas são calculadas levando em consideração a situação atual da simulação, sem considerar valores do passado distante.

5.3 Algoritmo de escalonamento desenvolvido

O algoritmo de escalonamento desenvolvido para avaliar as métricas trabalha da mesma forma para todas as métricas avaliadas e tem como objetivo equilibrar a carga dos processadores em relação à métrica utilizada.

Primeiramente, o algoritmo calcula a carga de cada processador. A carga de um processador é calculada pela soma das cargas dos processos alocados no processador em questão. Após calcular a carga de cada processador, o algoritmo calcula alguns valores que serão necessários para determinar se haverá ou não migração e quais processos serão migrados. Os valores calculados são os seguintes:

- Determinar os processadores mais e menos carregados;
- Determinar qual o **processo que será migrado**. O processo a ser migrado é o **processo mais lento do processador mais carregado**;
- Determinar a **diferença atual** de carga entre o processador mais carregado e o processador menos carregado. Determinar também a **diferença futura** entre esses dois processadores caso haja a migração do processo mais lento do processador mais carregado para o processador menos carregado. A diferença futura é uma estimativa e é calculada subtraindo a carga do processador mais carregado pela carga do processo a ser migrado, somando a carga do processo a ser migrado ao processador menos carregado e subtraindo a

carga estimada do processador originalmente mais carregado pela carga do processador originalmente menos carregado;

- Calcular a **média**, o **desvio padrão**, $\boldsymbol{\delta}$ e $\boldsymbol{\gamma}$ da carga dos processadores. Onde δ é a subtração da média pelo desvio padrão, enquanto γ é a soma da média pelo desvio padrão.

Calculados esses valores, resta determinar se haverá ou não uma migração. Uma migração será realizada se as duas condições a seguir forem verdadeiras:

- Se existir um processador cuja carga seja menor que δ ou existir um processador cuja carga seja maior que γ. Essa abordagem se baseia no algoritmo de balanceamento utilizando a taxa de avanço do LVT (BURDORF; MARTI, 1993), apresentado na Seção 3.5.2;
- Se as diferenças, atual e a estimativa futura, entre a carga dos processadores, mais e menos carregados, obedecerem uma das seguintes regras:
  - A diferença futura estimada ainda ser maior que zero, ou seja, mesmo retirando o processo mais lento do processador originalmente mais carregado e migrando para o processador originalmente menos carregado, o processador originalmente mais carregado ainda terá uma carga maior que a do processador originalmente menos carregado. Nesse caso, a migração é permitida pois o processador originalmente mais carregado está muito mais carregado que o processador originalmente menos carregado;
  - Caso a carga do processador originalmente menos carregado, após a estimativa de migração do processo mais lento, se torne maior que a carga do processo originalmente mais carregado, o quanto a carga do processador originalmente menos carregado pode ser maior que a carga do processador originalmente mais carregado não pode ser maior que a metade da diferença atual. Assim, pela estimativa, o processador originalmente mais carregado passará a ter uma carga menor que a carga

do processador originalmente menos carregado. Nesse caso, a migração só será realizada em casos onde a inversão entre os processadores mais e menos carregados leve a uma diferença que, segundo a estimativa, seja menor ou igual à metade da diferença atual.

Definido que haverá migração, o processo mais lento do processador mais carregado é migrado para o processador menos carregado. Essa é uma característica presente nos algoritmos de Carothers e Fujimoto (2000) e Reiher e Jefferson (1990). Conforme dito anteriormente, o algoritmo trabalha sempre da mesma forma. O que varia é a métrica usada para calcular a carga dos processos e, consequentemente, dos processadores.

A listagem 5.1 apresenta um pseudocódigo ilustrando os passos principais executados pelo algoritmo.

Listagem 5.1: Algoritmo de escalonamento utilizado para avaliação das métricas

```
1  void schedule(int* processMetricValue) {
2    bool shouldMigrate = false;
3    calculateMetricValueOnNodes(processMetricValue);
4
5    int loadedNode = findLoadedNode();
6    int unLoadedNode = findUnLoadedNode();
7
8    int nodesMean = calculateNodesMean();
9    int nodesStandardDeviation = calculateNodesStandardDeviation();
10
11   int deltaNodes = nodesMean - nodesStandardDeviation;
12   int gammaNodes = nodesMean + nodesStandardDeviation;
13
14   int differenceBetweenLoadedAndUnLoadedNodes = loadedNode -
        unLoadedNode;
15
16   int newDifferenceBetweenLoadedAndUnLoadedNodes = (loadedNode
17         - processMetricValue[loadedProcessFromLoadedNode])
```

```
18          - (unLoadedNode + processMetricValue[
                loadedProcessFromLoadedNode]);
19
20    if (newDifferenceBetweenLoadedAndUnLoadedNodes > 0)
21      shouldMigrate = true;
22    else {
23      int newAbsDiff = abs(newDifferenceBetweenLoadedAndUnLoadedNodes);
24
25      if (newAbsDiff <= (differenceBetweenLoadedAndUnLoadedNodes / 2))
26        shouldMigrate = true;
27    }
28
29    if ((unLoadedNode < deltaNodes || loadedNode > gammaNodes)
30        && (shouldMigrate))
31    {
32         migrateProcessTo(loadedProcessFromLoadedNode, unLoadedNodeName
               );
33    }
34  }
```

A carga média e o desvio padrão entre os processadores, aliados às diferenças, atual e estimada, entre a carga dos processadores, se mostraram eficientes em servir como condição de decisão para a migração. Em conjunto, esses fatores evitam que os processadores fiquem com cargas muito diferentes e evitam a migração desnecessária de processos, ou seja, aquelas migrações que causam pouco ou nenhum efeito no balanceamento da carga.

A cada migração, o algoritmo migra apenas um processo. Após uma migração, ou mesmo uma avaliação de carga que não resultou em migração, é aguardado um período de tempo, previamente configurado, até que uma nova avaliação de carga seja realizada. Optou-se pela migração individual de um processo para se fazer a menor quantidade de suposição possível em relação à carga da simulação, quando realizada a estimativa de diferenças entre os processadores. Isso é alcançado migrando apenas um processo a cada identificação de desbalanceamento, analisando o

efeito da migração na carga da simulação e tomando as próximas decisões baseadas em informações atualizadas que já refletem a nova distribuição dos processos.

O algoritmo resolve também o problema citado por Carothers e Fujimoto (2000) e Reiher e Jefferson (1990) em relação ao ajuste do fator de diferença entre as cargas que determina se haverá migração. Dessa forma, os valores da própria métrica são suficientes para a tomada de decisão, não necessitando de nenhuma informação externa à simulação.

5.4 Considerações Finais

Esse capítulo apresentou as métricas avaliadas e justificou suas escolhas, também mostrou como cada uma das métricas foi calculada.

O capítulo também introduziu uma nova métrica (EEW), mostrando como foi elaborada e como usá-la em um algoritmo de balanceamento.

Por fim, foi apresentado, em detalhes, o algoritmo de balanceamento dinâmico que foi utilizado para avaliação das métricas. Não foi identificada nenhuma característica no algoritmo proposto que possa beneficiar ou punir qualquer uma das métricas avaliadas, sendo considerado suficiente para a avaliação das métricas, que é uma das propostas deste trabalho.

O próximo capítulo apresenta os modelos, variações de carga e demais parâmetros utilizados nas simulações realizadas.

6 Modelos de simulação e variações de carga

O ambiente de execução permite que várias informações da simulação e também de controle do próprio ambiente sejam configuradas. A configuração dessas informações é que determina a estrutura do modelo, as variações de cargas e algumas informações de controle, como, por exemplo, o momento de encerrar a simulação. A variação de carga também pode ser produzida por carga externa, que não faz parte das configurações e é obtida na forma de um programa desenvolvido para sobrecarregar o processador.

Este capítulo descreve os parâmetros das simulações, o ambiente computacional onde foram executadas, os modelos de simulação utilizados e as variações de carga impostas. Apresenta também as simulações que foram executadas considerando todas as variações de ambiente configuradas.

6.1 Configurações dos parâmetros da simulação

As principais configurações utilizadas para realizar a modelagem da simulação, determinar parâmetros para variação de carga e ajustar as informações de controle do ambiente são as seguintes:

1. **Métrica de carga**: o ambiente permite que se defina qual das métricas será utilizada pelo algoritmo de balanceamento para a execução de uma

simulação;

2. **Intervalo de tempo entre cálculos de GVT**: determina a periodicidade com que o processo mestre irá realizar a solicitação de informações aos processos da simulação e também realizar o cálculo do GVT;

3. **Cálculos de GVT para migrar**: determina a periodicidade com que o algoritmo de balanceamento será executado. Essa periodicidade é determinada em relação aos cálculos de GVT. Por exemplo, uma configuração possível seria chamar o algoritmo de balanceamento a cada dois cálculos de GVT;

4. **Parâmetro de parada**: informa qual parâmetro será usado para encerrar a simulação. Os valores possíveis são o GVT e o número de eventos reais processados.

5. **Geração de eventos**: determina o padrão de geração de eventos dos processos, a probabilidade de geração de eventos a cada ciclo de execução do processo. Permite configurar um valor para cada processo;

6. **Tempo de serviço**: determina o tempo de serviço, tempo de processamento estimado para se processar um evento. Também permite que se configure um valor para cada processo;

7. **Probabilidade de mensagens**: determina quais os processos que se comunicam e com qual probabilidade. Configura o modelo da simulação e a frequência de envio de mensagens entre os processos.

8. **Quantidade de processos**: número de processos executando a simulação.

Algumas das configurações citadas foram ajustadas com o mesmo valor para todos os modelos e variações de carga testados. Outros parâmetros são justamente responsáveis pela configuração dos modelos e variação de carga e serão descritos nas seções 6.3 e 6.4, respectivamente.

Os valores para esses parâmetros em todas as simulações são os seguintes:

- **Métrica de carga**: cada uma das métricas listadas a Seção 5.1;
- **Intervalo de tempo entre cálculos de GVT**: 15 segundos;
- **Cálculos de GVT para migrar**: a cada 1 cálculo do GVT;
- **Parâmetro de parada**: 250.000 eventos reais processados;
- **Geração de eventos**: aleatório, conforme Seção 6.4;
- **Tempo de serviço**: aleatório, conforme Seção 6.4;
- **Probabilidade de mensagens**: varia de acordo com o modelo, Seção 6.3;
- **Quantidade de processos**: 25 processos;

6.2 *Cluster* de execução

O ambiente de execução foi composto por um *cluster* de cinco computadores. Em um dos computadores executava apenas o processo mestre que coordenava a simulação. Nos outros quatro computadores executavam os processos da simulação.

Todos os cinco computadores possuíam a seguinte configuração:

- **Processador**: *Intel Core2Quad* 2,66GHz;
- **Núcleos**: 4;
- **Cache**: L2:8MB;
- **Memória RAM**: 2 GB;
- **Placa de Rede**: *Ethernet* 100 Mbps.
- **Arquitetura**: 32 *bits*.
- **Sistema operacional**: *Linux Fedora* 13;

6.3 Modelos

As simulações foram realizadas com três modelos diferentes. Cada modelo apresenta diferenças entre a conectividade dos processos, com quais processos um determinado processo se comunica, e a probabilidade de troca de mensagens, qual a frequência de envio de mensagens de um processo em relação aos processos com os quais se comunica. É considerado que dois processos, P_1 e P_2, se comunicam quando P_1 envia mensagens para P_2 e/ou P_1 recebe mensagens enviadas por P_2.

6.3.1 Modelo de grupos

No modelo de grupos, os processos são divididos em quatro grupos, cada grupo de processos com características diferentes. São eles:

- **Grupo vermelho**: 7 processos;
- **Grupo verde**: 6 processos;
- **Grupo amarelo**: 6 processos;
- **Grupo azul**: 6 processos.

Os grupos, vermelho e verde, se comunicam da seguinte forma: os processos do grupo vermelho não enviam mensagens a nenhum processo do grupo verde, enquanto cada processo do grupo verde envia mensagens a todos os processos do grupo vermelho a uma frequência de 1% para cada processo com o qual se comunica. Isso significa que, em um dado intervalo de tempo, suficiente para geração de 100 eventos em cada processo, os processos do grupo vermelho processam 106 eventos cada, enquanto os processos do grupo verde processam 93 eventos cada, o que faz com que os processos do grupo vermelho tenham uma carga de 57%, enquanto os processos do grupo verde tenham uma carga de 43%, ou seja, os

Grupo	Carga de eventos	Carga de mensagens
Vermelho	30%	17%
Verde	22%	0%
Amarelo	25%	50%
Azul	23%	33%

Tabela 1: Carga e recebimento de mensagens de um processo de cada grupo em relação a toda a carga

processos do grupo vermelho possuem uma carga 32,5% maior em relação aos processos do grupo verde, além de receber mensagens dos processos do grupo verde, o que pode levar a erros de causa e efeito. Nenhum dos grupos, vermelho ou verde, se comunica com qualquer outro grupo da simulação.

Os grupos, amarelo e azul, se comunicam da seguinte forma: os processos do grupo amarelo enviam mensagens a todos os processos do grupo azul a uma frequência de 2% para cada processo com o qual se comunica. Os processos do grupo azul enviam mensagens a todos os processos do grupo amarelo a uma frequência de 3% para cada processo com o qual se comunica. Sendo assim, em um dado intervalo de tempo, suficiente para geração de 100 eventos em cada processo, os processos do grupo amarelo processam 106 eventos cada, enquanto os processos do grupo azul processam 94 eventos cada, o que faz com que os processos do grupo amarelo tenham uma carga de 53%, enquanto os processos do grupo azul tenham uma carga de 47%, ou seja, os processos do grupo amarelo possuem uma carga 12,7% maior em relação aos processos do grupo azul. Além de estarem mais carregados, os processos do grupo amarelo recebem 50% mais mensagens do que os processos do grupo azul, portanto, estão mais sujeitos a sofrer *rollbacks*.

Considerando o mesmo intervalo de tempo, suficiente para geração de 100 eventos em cada processo da simulação, a Tabela 1 apresenta a porcentagem de carga e de recebimento de mensagens de um processo de cada grupo em relação a toda a carga.

As diferenças na carga de eventos para os grupos, menos e mais carregados, variam em 36,4%, aproximadamente. Em relação às mensagens, enquanto os processos do grupo verde não recebem mensagens, 50% das mensagens enviadas têm como destinos os processos do grupo amarelo.

A Figura 10 ilustra o modelo de grupos.

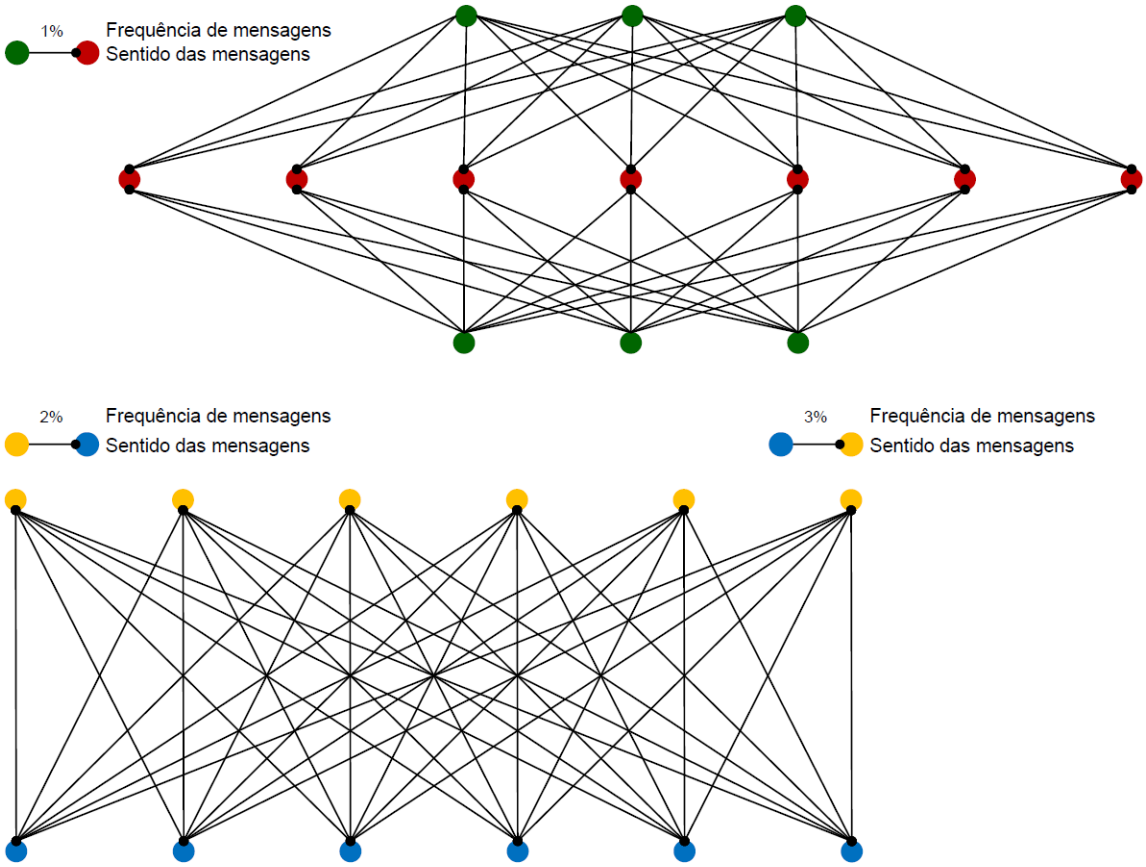

Figura 10: Modelo de grupos

6.3.2 *Hierarchical Network Model - hnet*

Baseado no modelo usado por Glazer e Tropper (1993), esse modelo organiza os processos em quatro níveis hierárquicos, nível 01, nível 02, nível 03 e nível 04. Há geração de eventos em todos os níveis e os processos podem enviar mensagens tanto

para o nível superior quanto para o nível inferior. Nessa estrutura hierárquica, há um único caminho entre quaisquer pares de processos.

A Figura 11 ilustra o modelo hierárquico e apresenta a quantidade de processos em cada nível.

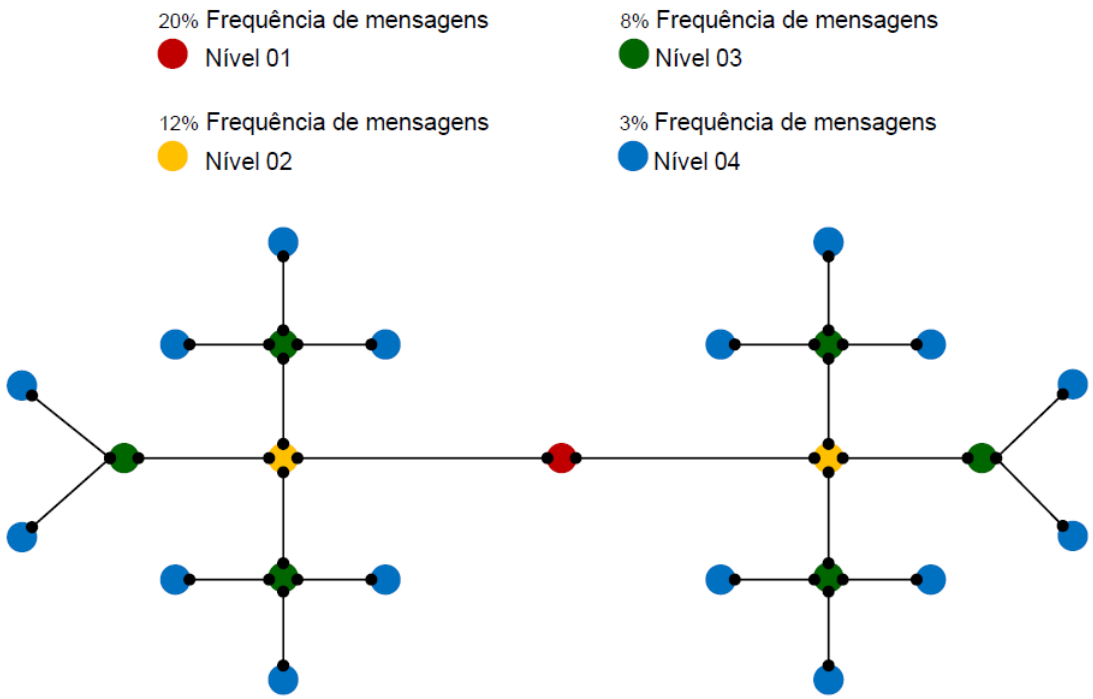

Figura 11: Modelo hierárquico

A frequência de envio de mensagens está relacionada ao nível que o processo pertence. 20% dos eventos nos processos de nível 01 geram eventos para os processos do nível 02. 12% dos eventos nos processos de nível 02 geram eventos para os processos do nível 01 e para os processos do nível 03. 8% dos eventos nos processos de nível 03 geram eventos para os processos do nível 02 e para os processos do nível 04 e 3% dos eventos nos processos de nível 04 geram eventos para os processos do nível 03.

A frequência de envio de mensagens de um processo é dividida entre os processos com os quais este processo se comunica. Por exemplo, os 20% de frequência do processo de nível 01 é dividido entre os dois processos do nível 02, ou seja, 10%

para cada processo. Para os níveis centrais, cujos processos se comunicam tanto com o nível superior quanto com o nível inferior, a divisão é feita da mesma forma, independente dos processos estarem no nível superior ou inferior. Por exemplo, os 12% de frequência dos processos de nível 02 são divididos entre o processo de nível 01 e os processos de nível 03, com 3% para cada um dos quatro.

6.3.3 *Distributed Network Model - dnet*

Também baseado no modelo usado por Glazer e Tropper (1993), esse modelo simula uma rede de computadores, podendo ser comparado aos roteadores da *Internet*, que conectam sub-redes e *hosts*. Os processos são divididos em 5 regiões. Uma região central conecta as outras 4 regiões.

A frequência de mensagens é a mesma para todos os processos e é uniformemente dividida entre os processos comunicantes. No modelo proposto, 6% dos eventos processados por cada processo geram mensagens para os processos comunicantes. A taxa de mensagens que um processo pode receber depende de com quantos processos cada processo se comunica.

A Figura 12 ilustra a estrutura do modelo distribuído.

A frequência de envio de mensagens de um processo, 6% para todos os processos desse modelo, é dividida uniformemente entre seus processos comunicantes. Por exemplo, um processo que se comunica com 3 processos terá um frequência de envio de 2% para cada processo, enquanto um processo que se comunica com 2 processos terá uma frequência de envio de 3% para cada evento. Em relação ao recebimento de mensagens, quanto maior for o número de processos que um determinado processo se comunica, maior será sua taxa de recebimento de mensagens.

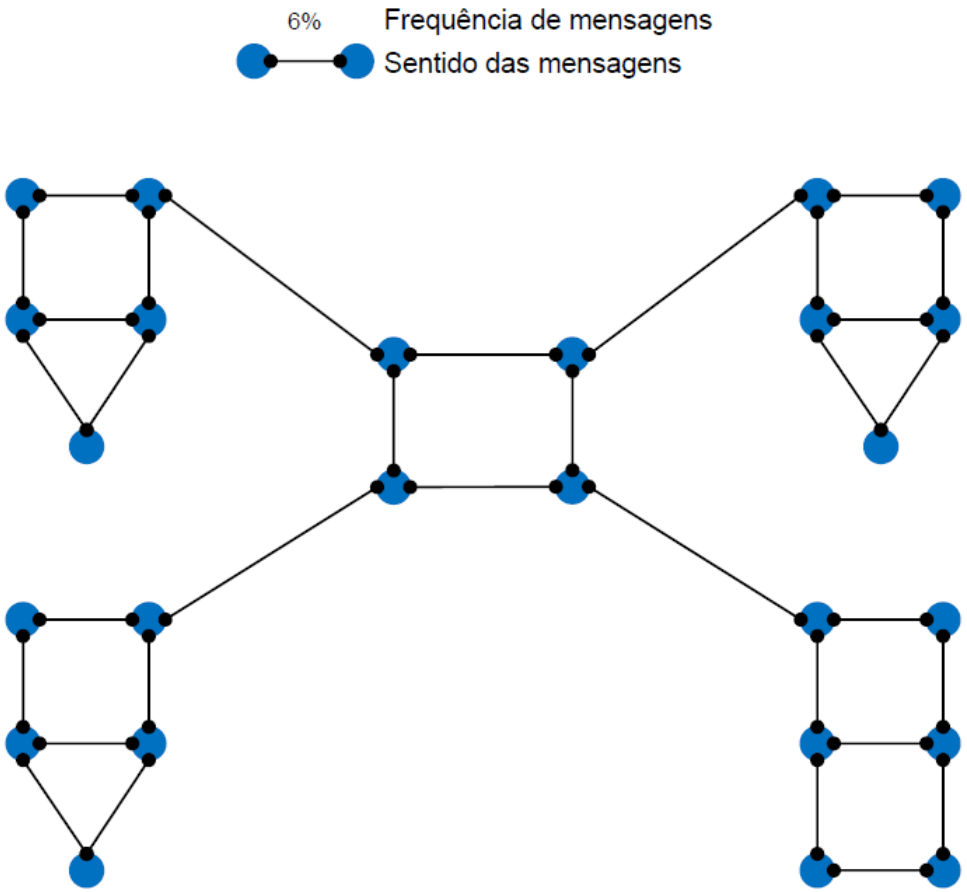

Figura 12: Modelo distribuído

6.4 Variações de carga

A carga imposta a cada processo da simulação é afetada pela variação de três fatores: (1) a probabilidade de geração de eventos, (2) o tempo gasto para o processamento de cada evento e (3) cargas externas impostas por um programa externo à simulação. Essa variação de carga foi baseada nos fatores de desbalanceamento de carga levantados por Carothers e Fujimoto (2000), Seção 3.4.

A probabilidade de geração de eventos e o tempo gasto para o processamento de cada evento são determinados de maneira aleatória no momento da criação de cada processo. Os valores possíveis para cada um desses parâmetros são os seguintes:

- **Probabilidade de geração de eventos**: a probabilidade de geração de eventos de um processo é aleatoriamente determinada dentre um dos seguintes valores: 40%, 60% ou 80%. Isso significa que, a cada ciclo do processo, são esses os valores que indicam a probabilidade de geração de um novo evento.

- **Tempo gasto para o processamento de cada evento**: o tempo gasto para o processamento de cada evento é aleatoriamente determinado dentre um dos seguintes valores: 10, 15 ou 20. Isso significa que, ao executar um processo, são esses os valores que determinam o tempo de processamento de cada evento.

Os parâmetros citados acima são determinados no início da execução do processo e não mudam durante a simulação. Por exemplo, se ao iniciar o processo P_1 foi determinado que sua probabilidade de geração de eventos é x e seu tempo de serviço é y, esses valores serão os mesmos até o final da execução da simulação.

A mudança desses parâmetros acontece a cada execução de uma nova simulação. Conforme será explicado na seção 6.5, para cada métrica e para cada modelo foram executadas 10 simulações. Cada uma dessas simulações possuem sementes de aleatoriedade diferentes, ou seja, para uma mesma métrica e um mesmo modelo, foram executadas 10 simulações com variação de carga diferentes, baseadas na aleatoriedade da probabilidade de geração de eventos e tempo de serviço de cada processo.

A variação do tempo de serviço dos processos implica que, para um processo com tempo de serviço igual a 20, o tempo de execução de cada evento desse processo será duas vezes maior do que o tempo de execução dos eventos de um processo com o tempo de serviço igual a 10. Dentro do ambiente de execução das simulações, isso foi alcançado fazendo com que uma atividade fosse realizada quantas vezes fosse o valor do parâmetro durante a execução de cada evento. Por exemplo, para um processo com tempo de serviço igual a 10, a cada evento processado, uma atividade de processamento, que no caso foi uma multiplicação de matrizes, é executada 10

vezes.

A carga externa consiste em um programa desenvolvido apenas para consumir recursos de processamento do ambiente. O programa inicializa algumas *threads*, as quais executam uma multiplicação de matrizes. O programa consome mais ou menos poder de processamento dependendo da quantidade de *threads* realizando as multiplicações. A quantidade de *threads* a ser executadas é passada como parâmetro para o programa.

Em relação à carga externa, os quatro computadores que executaram as simulações possuíam as seguintes configurações:

- **Computador 1**: processador com aproximadamente 75% da capacidade dedicada à simulação, com os outros 25% da capacidade sendo consumida por carga externa;
- **Computador 2**: processador com aproximadamente 50% da capacidade dedicada à simulação, com os outros 50% da capacidade sendo consumida por carga externa;
- **Computador 3**: processador totalmente dedicado à simulação;
- **Computador 4**: processador totalmente dedicado à simulação.

Essa configuração de carga faz com que haja uma diferença mínima de 33% e uma diferença máxima de 100% entre as capacidades de processamento dedicadas à simulação. Essa configuração de carga externa se manteve fixa para todas as métricas e para todos os modelos.

6.5 Simulações realizadas

As simulações realizadas foram executadas com cada um dos modelos apresentados, combinados com as variações de carga aleatoriamente determinadas e

também com as métricas utilizadas pelo algoritmo de balanceamento. Por exemplo, para o modelo *hnet* e métrica EEW, foram executadas 10 simulações. Cada uma dessas 10 simulações possui uma semente diferente.

Essa abordagem se repetiu para cada um dos modelos (grupos, *hnet* e *dnet*) e para cada uma das métricas (taxa de avanço do LVT, utilização efetiva, utilização efetiva baseada em ciclos, PAT e EEW). Para cada um dos modelos, também foram executadas 10 simulações com o algoritmo de balanceamento *Round Robin*. Os dados nas execuções realizadas com o *Round Robin* foram utilizados para realizar a análise do desempenho do algoritmo de balanceamento proposto. No ambiente, o algoritmo de *Round Robin* divide os processos igualmente entre os processadores disponíveis, essa foi também a distribuição inicial dos processos para o algoritmo de balanceamento proposto. Ou seja, inicialmente, quando ainda não se tem nenhuma informação sobre a execução da simulação, o *Round Robin* foi usado como distribuição inicial.

6.6 Considerações Finais

Esse capítulo apresentou todas as informações necessárias para compreender como as simulações foram realizadas, evidenciando todos os parâmetros importantes, tais como, informações dos modelos, do ambiente e das variações de carga. Essas informações também são importantes para que se possa reproduzir os experimentos, caso necessário.

O próximo capítulo apresenta uma análise sobre o desempenho do algoritmo de escalonamento e das métricas avaliadas.

7 Análises dos resultados

As análises dos resultados obtidos com a execução das simulações realizadas foram feitas com base em três fatores: (1) a eficiência, (2) o número de migrações realizadas e (3) o tempo total de execução da simulação.

A **eficiência** é a relação entre o número de eventos reais processados, aqueles que não sofreram *rollbacks*, pelo número total de eventos processados. Por exemplo, se em uma simulação foram executados 100 eventos no total, dos quais 60 sofreram *rollbacks*, então a eficiência dessa simulação foi de 40%.

O **número de migrações** realizadas indica quantos processos sofreram migrações no decorrer da execução da simulação e o **tempo total de execução** determina o tempo real (em segundos) gasto para executar a simulação.

A análise foi feita em relação a esses três fatores devido ao fato da eficiência não ser diretamente afetada pelo número de migrações realizadas, o que já pode ocorrer em relação ao tempo de execução, uma vez que a ação de migrar processos consome recursos de processamento e tempo.

Foi realizada uma análise individual sobre cada modelo utilizado e também uma análise geral envolvendo os três modelos. Os três fatores utilizados nas análises são comparados ao algoritmo estático utilizado na distribuição inicial dos processos, o *Round Robin*. Os valores apresentados nos gráficos são a média das 10 execuções realizadas para cada modelo com cada uma das métricas. O desvio padrão também é exibido no gráfico, a fim de mostrar o quanto houve de variação dos fatores entre as execuções. O percentual de melhora de cada métrica também é exibido.

7.1 Análise de desempenho com o modelo *dnet*

Conforme pode ser visto no gráfico da Figura 13, a métrica que alcançou a melhor eficiência foi a taxa de avanço do LVT, com aproximadamente 13% de melhora. Em relação à métrica PAT, que alcançou a menor melhora na eficiência, aproximadamente 7%, a melhora alcançada pela taxa de avanço do LVT foi, aproximadamente, 88% maior. A melhora de eficiência das outras métricas estão próximas a média de todas as métricas. A melhora na eficiência alcançada pela taxa de avanço do LVT foi aproximadamente 23% maior em relação a média de todas as métricas, que foi de aproximadamente 11%, sendo que a métrica PAT teve uma melhora aproximadamente 31% abaixo da média. O algoritmo de balanceamento, considerando a média de todas as métricas, obteve uma melhora na eficiência de aproximadamente 11% em relação ao *Round Robin*.

Para esse modelo, o tempo de execução das simulações não refletiu diretamente na eficiência, ou seja, as métricas mais eficientes não foram as que obtiveram os menores tempos de execução das simulações, conforme pode ser visto no gráfico da Figura 14. Por exemplo, as duas métricas mais eficientes, a taxa de avanço do LVT e a utilização efetiva, obtiveram o segundo e terceiro maior tempo de execução, respectivamente. No entanto, as diferenças entre o tempos de execução de cada métrica são muito pequenos, com exceção da métrica PAT, que obteve uma redução no tempo de execução de aproximadamente 82% abaixo da média das métricas. Em média, o algoritmo de balanceamento obteve uma redução de aproximadamente 5% no tempo de execução das simulações. Já em relação a eficiência, a média de melhora foi de aproximadamente 11%.

Para melhor entender esse cenário, é importante analisar também o número de migrações realizadas por cada métrica. Essas informações podem ser vistas no gráfico da Figura 15.

Em relação ao número de migrações, houve uma diferença grande entre a métrica que mais migrou processos, a taxa de avanço do LVT, com uma média superior

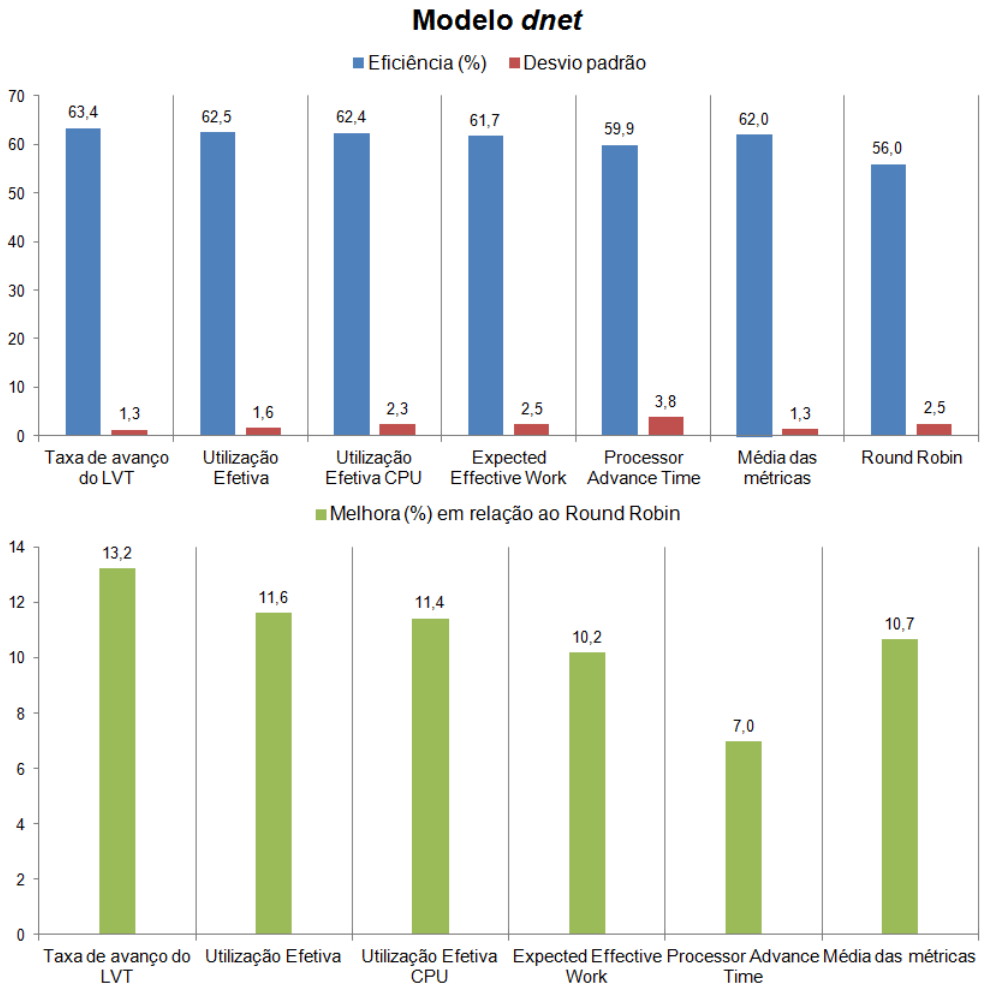

Figura 13: Eficiência das métricas no modelo *dnet*

a 5 migrações por execução, e a métrica que menos migrou processos, a métrica EEW, com uma média inferior a 2 migrações por execução. A diferença entre a métrica que menos migrou e a métrica que mais migrou chega a aproximadamente 178%, enquanto a diferença entre a métrica que mais migrou ficou aproximadamente 60% acima da média das métricas e a diferença entre a métrica que menos migrou ficou aproximadamente 73% abaixo da média das métricas. Um outro fator interessante, em relação ao número de migrações para esse modelo, é que os desvios padrão são relativamente altos, principalmente em relação à métrica da taxa de avanço do LVT.

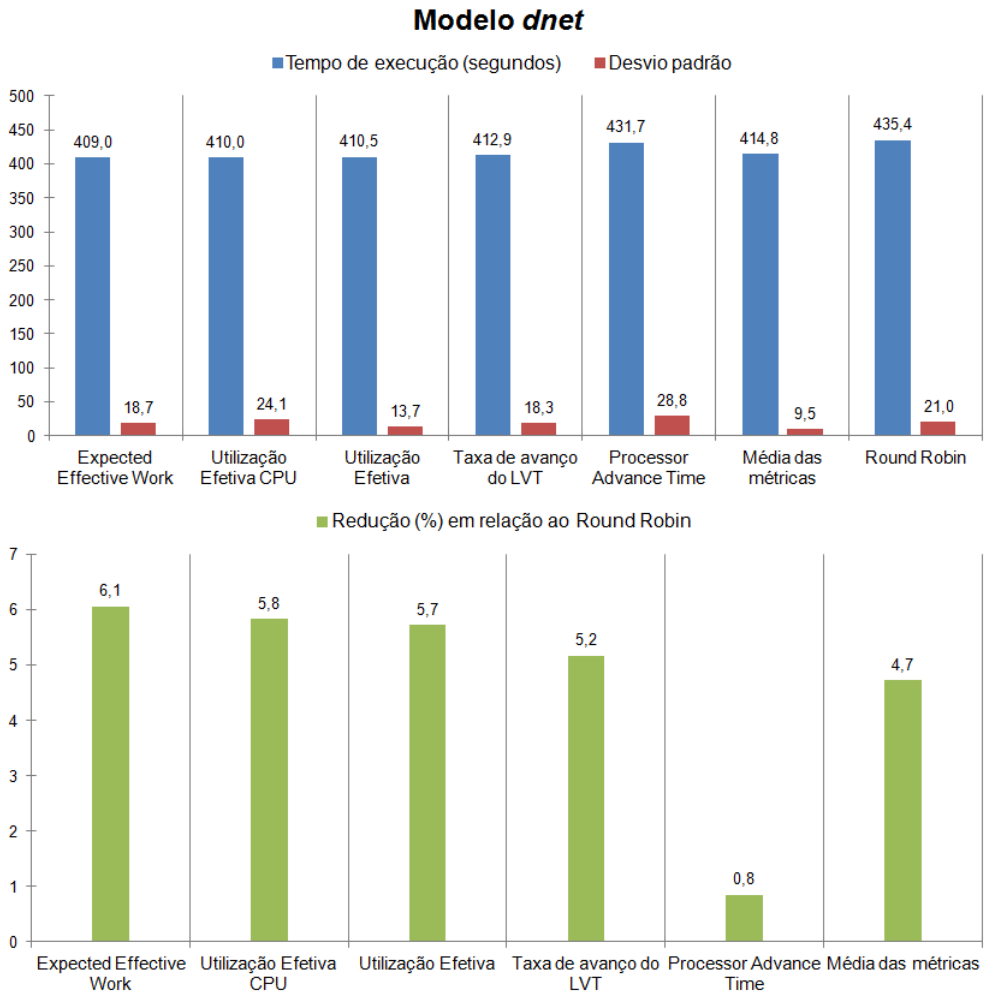

Figura 14: Tempo de execução para cada métrica no modelo *dnet*

Tomando como base esse modelo, observa-se algumas características em relação à eficiência, o número de migrações e o tempo de execução. Um número de migrações alto nem sempre leva a uma alta eficiência. Pelo modelo, isso pode ser observado pela métrica PAT, a segunda que mais migra e a pior em eficiência. Já um número de migrações mais baixo leva a uma redução no tempo de execução da simulação, considerando que a métrica tenha uma boa eficiência, como pode ser visto em relação à métrica EEW. Se o número de migrações necessário para alcançar uma boa eficiência for alto, uma boa eficiência pode não levar a uma boa redução do tempo de execução, como pode ser visto em relação à métrica da taxa

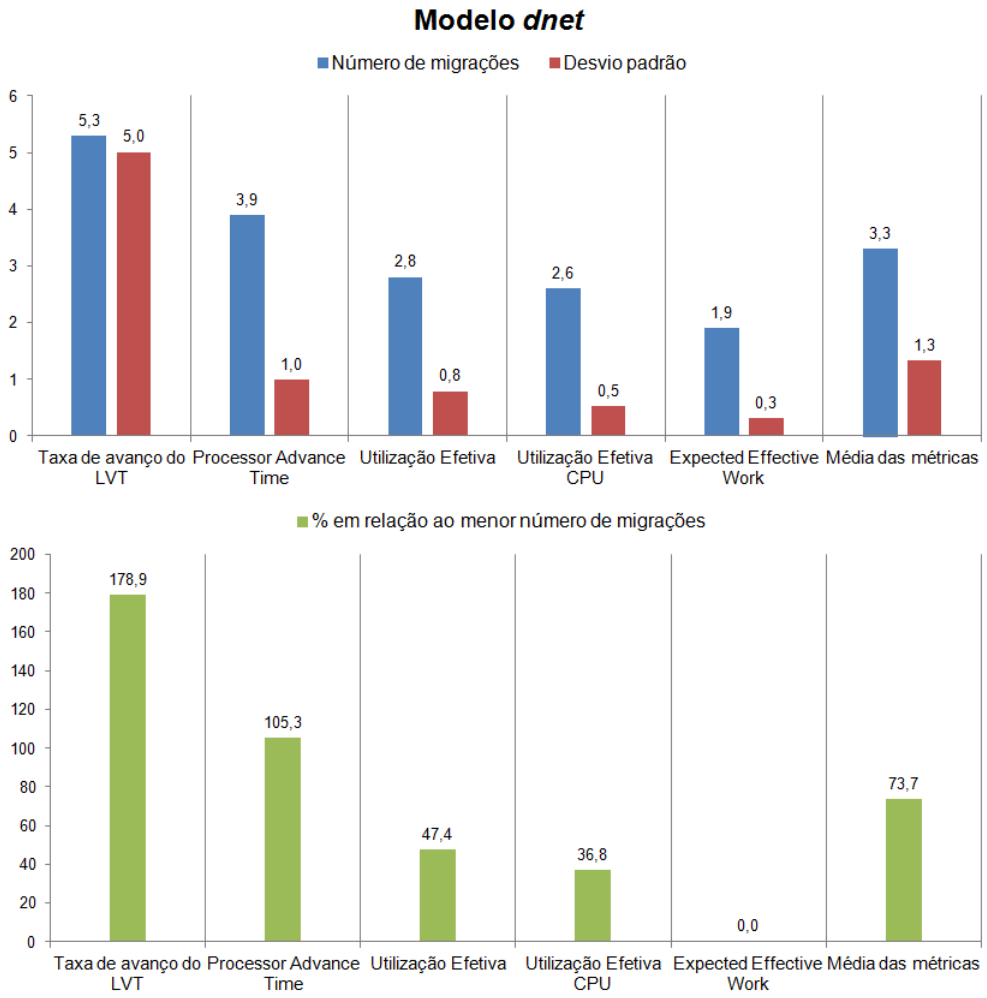

Figura 15: Número de migrações para cada métrica no modelo *dnet*

de avanço do LVT.

Verifica-se que não há uma relação direta entre a eficiência e a redução no tempo de execução, ou seja, as métricas mais eficientes não necessariamente são as que mais reduzem o tempo de execução, isso, além de outros fatores, devido ao número de migrações executadas. As métricas que conseguem a maior redução no tempo de execução são aquelas que conseguem um bom equilíbrio entre eficiência e número de migrações, como pode ser visto em relação à métrica EEW e às métricas baseadas na utilização efetiva.

7.2 Análise de desempenho com o modelo de grupos

Os resultados obtidos para o modelo de grupos são consideravelmente diferentes dos resultados obtidos para o modelo *dnet*. Conforme pode ser visto no gráfico da Figura 16, a métrica PAT, que obteve a menor eficiência no modelo *dnet*, obteve a melhor eficiência para o modelo de grupos, obtendo uma melhora aproximadamente 44% maior que a métrica EEW, que obteve a menor eficiência em relação ao modelo de grupos, e uma melhora aproximadamente 16% maior que a média das métricas. Em relação a métrica da taxa de avanço do LVT, a melhora na eficiência da métrica PAT foi de aproximadamente 19%, bem menor em relação aos 88% alcançados pela taxa de avanço do LVT no modelo *dnet*.

Observa-se uma oscilação grande entre a eficiência alcançada pela métrica PAT em relação ao modelo *dnet* (7%) e o modelo de grupos (15%), uma variação de aproximadamente 114%, enquanto as outras métricas oscilam bem menos. Por exemplo, a segunda métrica que mais oscilou foi a métrica da utilização efetiva calculada com base em ciclos de CPU, com uma variação de aproximadamente 25%.

Mais uma vez, a eficiência não refletiu diretamente na redução do tempo de execução das simulações. Conforme pode ser visto no gráfico da Figura 17, a métrica mais eficiente, a métrica PAT, obteve apenas o terceiro melhor tempo de execução, que, a propósito, foi praticamente o mesmo em relação ao *Round Robin*, ou seja, apesar de uma eficiência 15% melhor, praticamente não houve melhora no tempo de execução. Dois casos ainda piores são os das métricas baseadas na utilização efetiva que, apesar de ter alcançado eficiências superiores à média, obtiveram tempos de execução maiores que o tempo de execução do *Round Robin*. Por outro lado, a métrica da taxa de avanço do LVT, com uma eficiência próxima a média de todas as métricas, obteve o menor tempo médio de execução.

O gráfico da Figura 18 reforça que o número de migrações afeta a relação da eficiência com o tempo de execução. Apesar das três métricas que mais migraram,

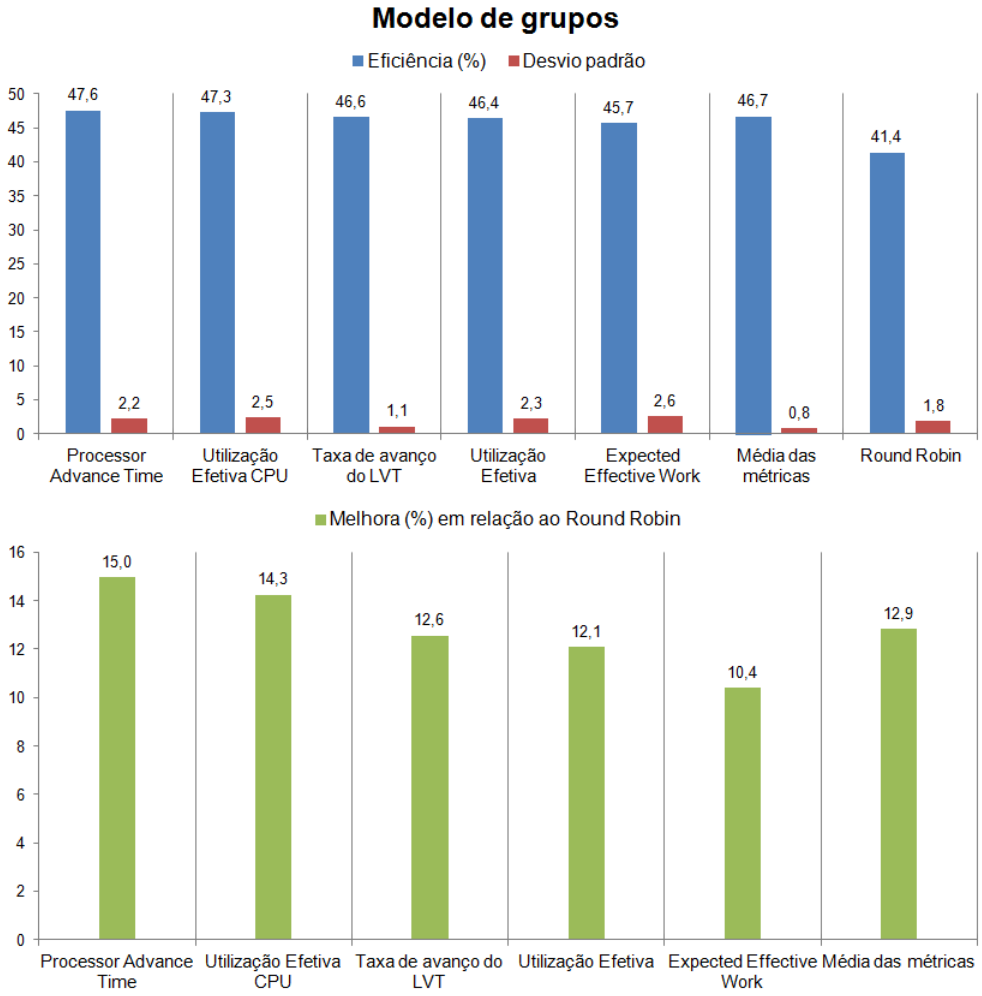

Figura 16: Eficiência das métricas no modelo de grupos

a métrica PAT e as duas métricas baseadas em utilização efetiva, terem obtido os maiores valores de eficiência, elas obtiveram as piores valores em relação à redução do tempo de execução. Já a métrica EEW, que migrou menos e obteve a menor eficiência para esse modelo, ainda obteve a segunda melhor redução em relação ao tempo de execução.

Em média, para o modelo de grupos, o algoritmo de balanceamento obteve uma eficiência 13% melhor que o algoritmo *Round Robin*. Esse valor foi superior à melhora alcançada pelo algoritmo no modelo *dnet*. No entanto, enquanto a redução

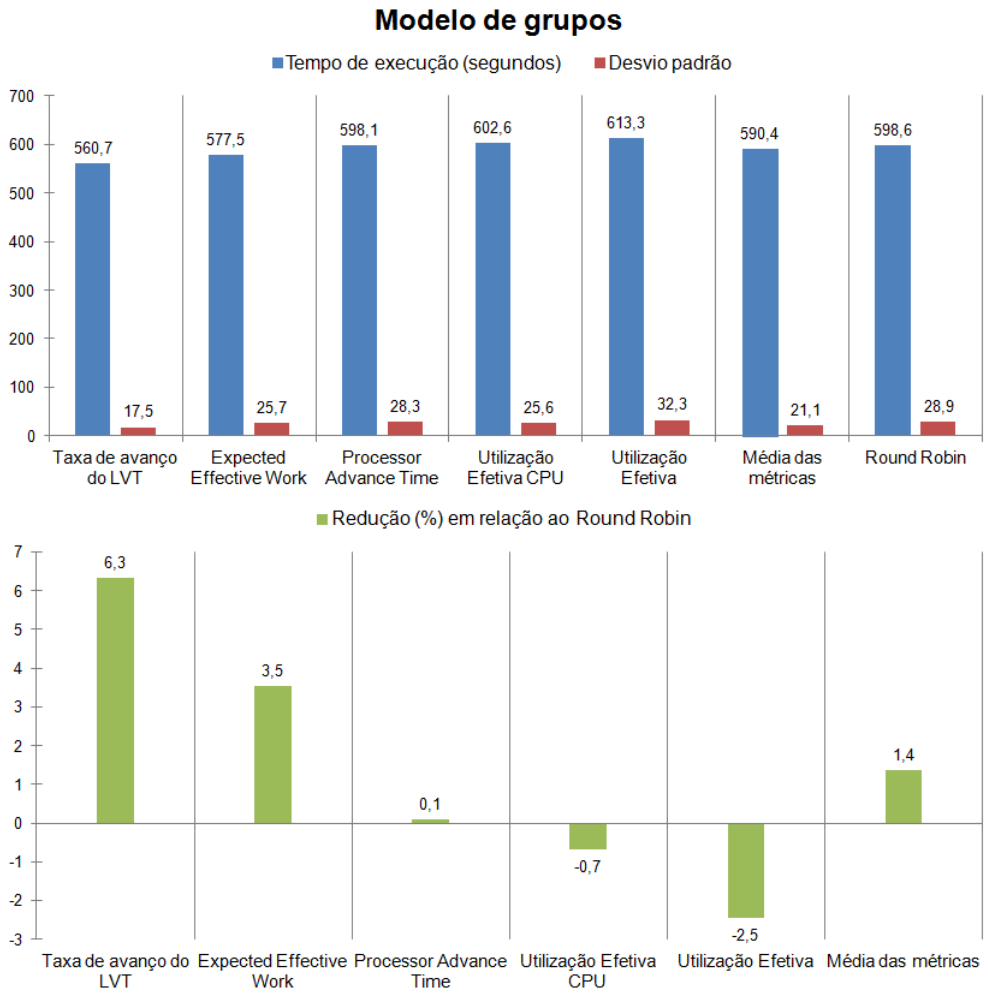

Figura 17: Tempo de execução para cada métrica no modelo de grupos

média no tempo de execução alcançada no modelo *dnet* foi de aproximadamente 5%, a redução média no tempo de execução para o modelo de grupos chegou a apenas 1,5%. Essa redução menor no tempo de execução é consequência de um número médio de migração maior para o modelo de grupos.

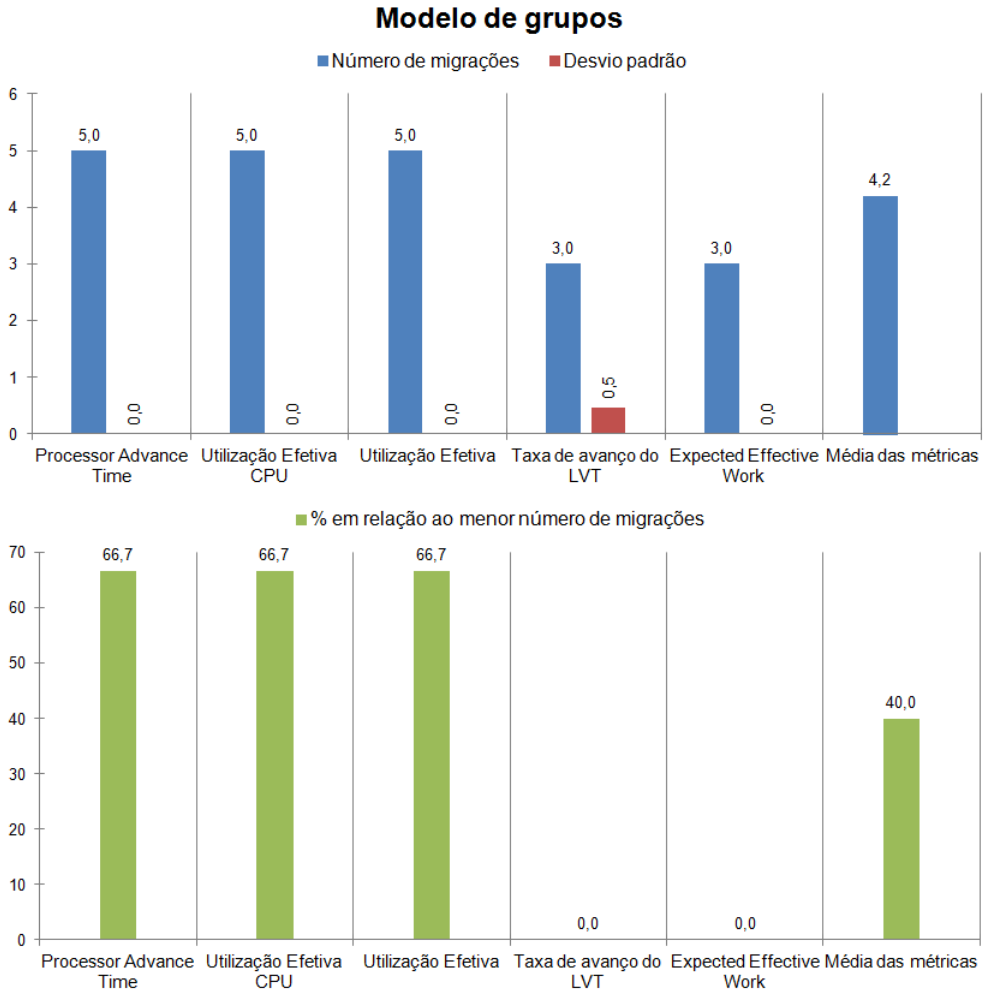

Figura 18: Número de migrações para cada métrica no modelo de grupos

7.3 Análise de desempenho com o modelo *hnet*

Conforme pode ser observado no gráfico da Figura 19, a métrica da utilização efetiva obteve a maior eficiência para o modelo *hnet*. Nota-se que, para cada um dos modelos, uma métrica diferente obteve a melhor eficiência. A métrica PAT volta a ser a que obteve a menor eficiência, com a métrica da utilização efetiva possuindo uma melhora na eficiência aproximadamente 47% superior. As outras métricas mantém um certo equilíbrio na melhora da eficiência em relação aos outros modelos.

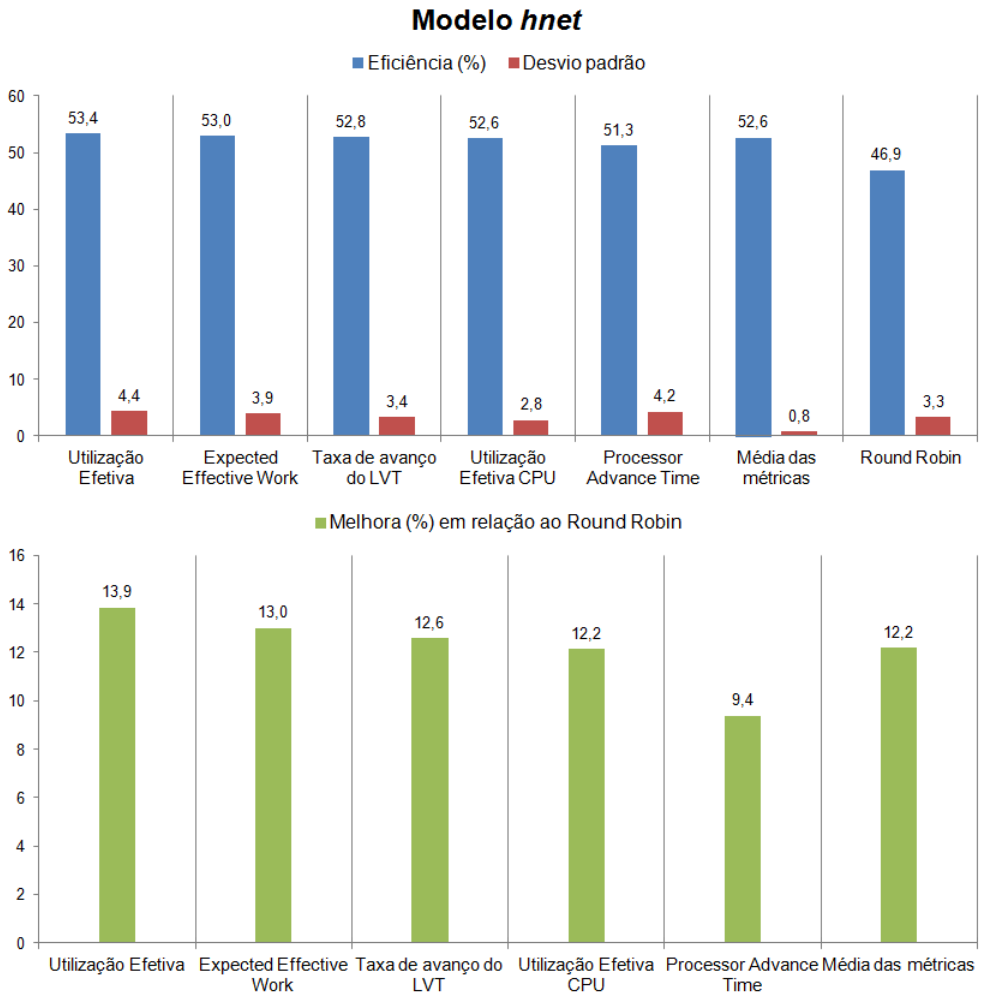

Figura 19: Eficiência das métricas no modelo *hnet*

A análise desse terceiro modelo comprova que a eficiência não reflete diretamente, para todos os casos, na redução do tempo de execução das simulações. Conforme pode ser visto no gráfico da Figura 20, a métrica mais eficiente, a métrica da utilização efetiva, obteve a segunda pior redução no tempo de execução da simulação, enquanto a métrica da taxa de avanço do LVT, que obteve apenas a terceira melhor eficiência, obteve novamente a melhor redução no tempo de execução das simulações.

Com a análise desse terceiro modelo fica evidente que a não relação entre a

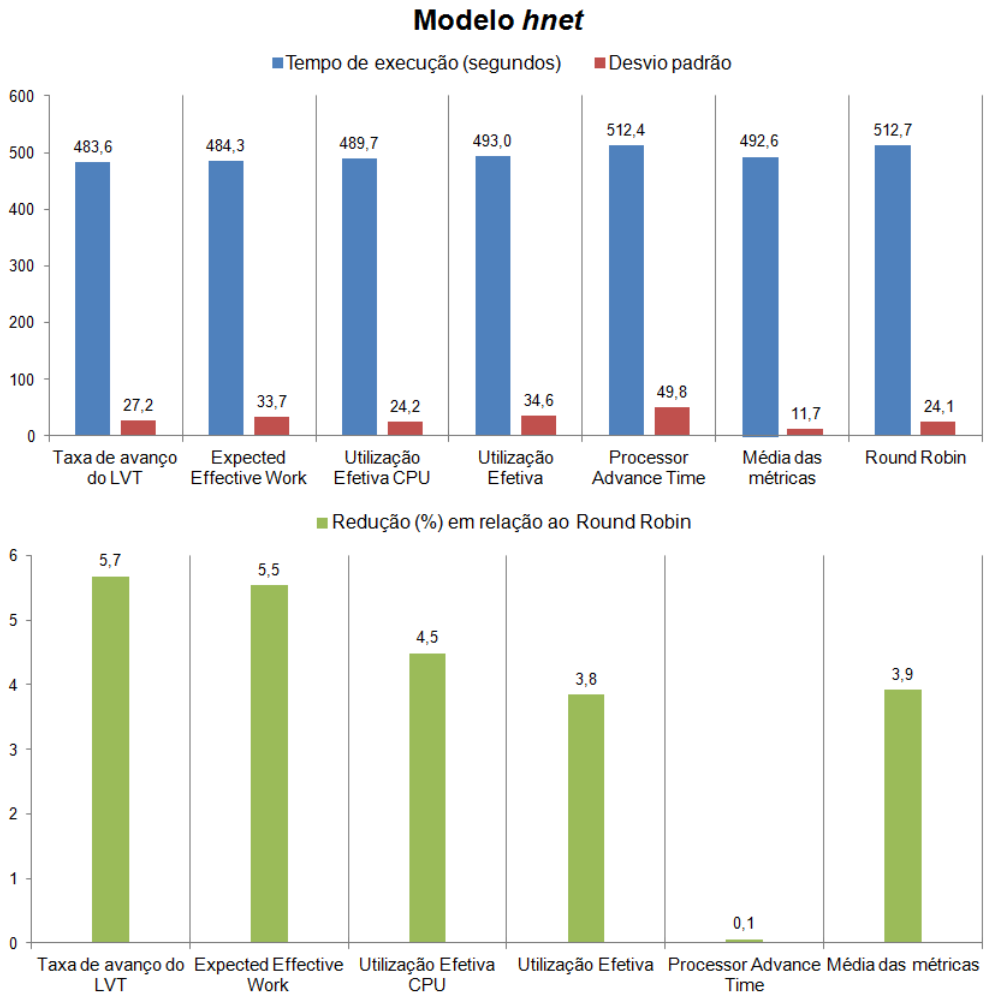

Figura 20: Tempo de execução para cada métrica no modelo *hnet*

eficiência e a redução no tempo de execução é devido ao número de migrações realizadas. Conforme pode ser visto no gráfico da Figura 21, as métricas que mais migraram, mesmo tendo uma alta eficiência, são as que menos reduzem o tempo de execução.

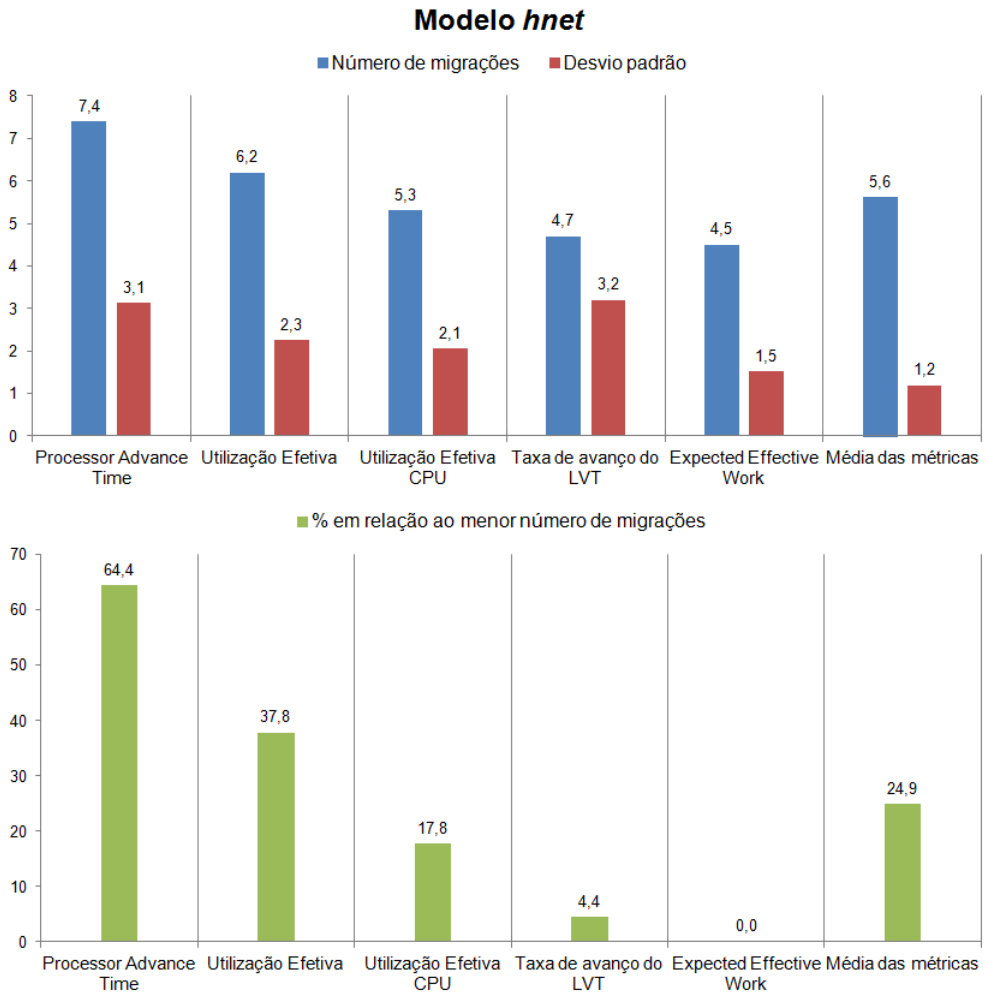

Figura 21: Número de migrações para cada métrica no modelo *hnet*

7.4 Análise de desempenho em relação a todos os modelos

O gráfico da Figura 22 mostra a média da eficiência para cada métrica em relação aos três modelos. A métrica da taxa de avanço do LVT foi a mais eficiente considerando todos os modelos, com uma eficiência 13% maior que o *Round Robin*, uma melhora na eficiência de aproximadamente 18% em relação a métrica menos eficiente, a métrica PAT, e uma melhora na eficiência de aproximadamente 8% em relação à média de todas as métricas. A segunda métrica mais eficiente foi a métrica da utilização efetiva, com a mesma eficiência para a utilização calculada

pelo número de eventos cancelados quanto para a utilização calculada baseada em ciclos de CPU. Essas três métricas citadas ficaram acima da média de eficiência de todas as métricas, já as métricas EEW e PAT, ficaram abaixo da média, com a métrica EEW sendo superior à métrica PAT.

De uma maneira geral, o algoritmo de balanceamento implementado teve uma eficiência média, considerando todas as métricas, aproximadamente 12% superior ao *Round Robin*.

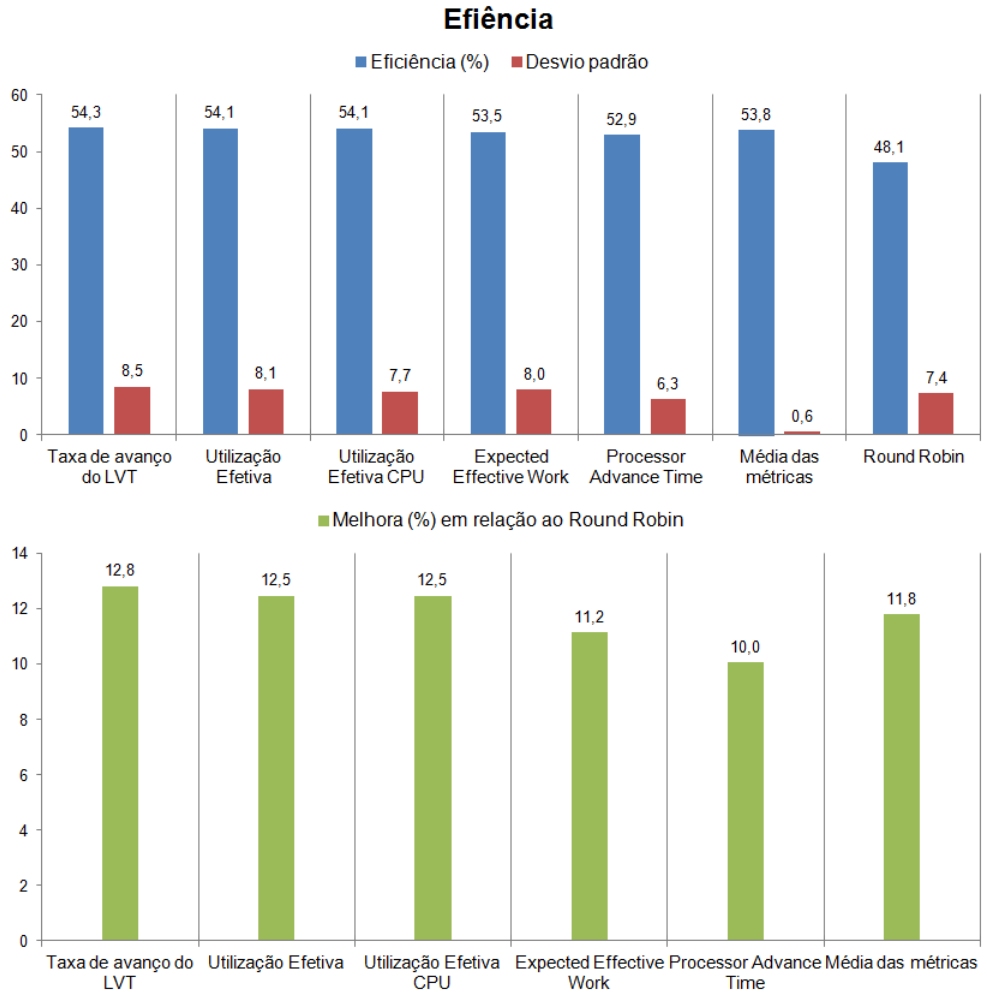

Figura 22: Eficiência das métricas em relação aos três modelos (grupos, *hnet* e *dnet*)

Em relação ao número de migrações, o que pode ser verificado pelo gráfico da

Figura 23 é que, conforme citado anteriormente, migrar muito não é sinal de uma melhor eficiência. As métricas que conseguiram as melhores eficiências tiveram o número de migrações bem próximos à média de migrações de todas as métricas.

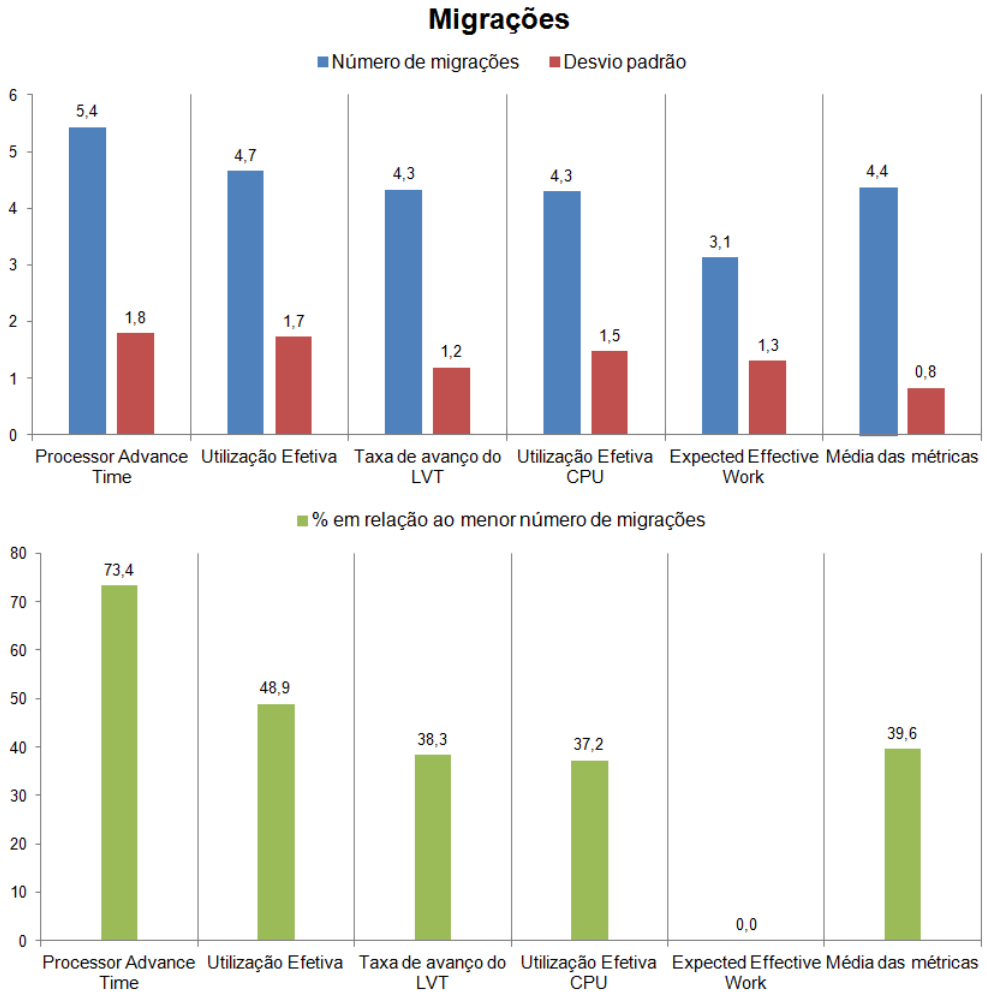

Figura 23: Número de migrações para cada métrica em relação aos três modelos (grupos, *hnet* e *dnet*)

Pelo gráfico da Figura 24, pode ser observado que o número de migrações tem influência na redução do tempo de execução. Isso pode ser notado considerando as métricas que menos migram, métrica EEW, taxa de avanço do LVT e utilização efetiva, e seus respectivos tempos de execução. Enquanto a métrica da taxa de avanço do LVT obteve uma redução de aproximadamente 6% no tempo de execu-

ção, a métrica PAT praticamente não reduziu o tempo de execução. A média de redução do tempo de execução para todas as métricas foi de apenas 3% em relação ao *Round Robin*.

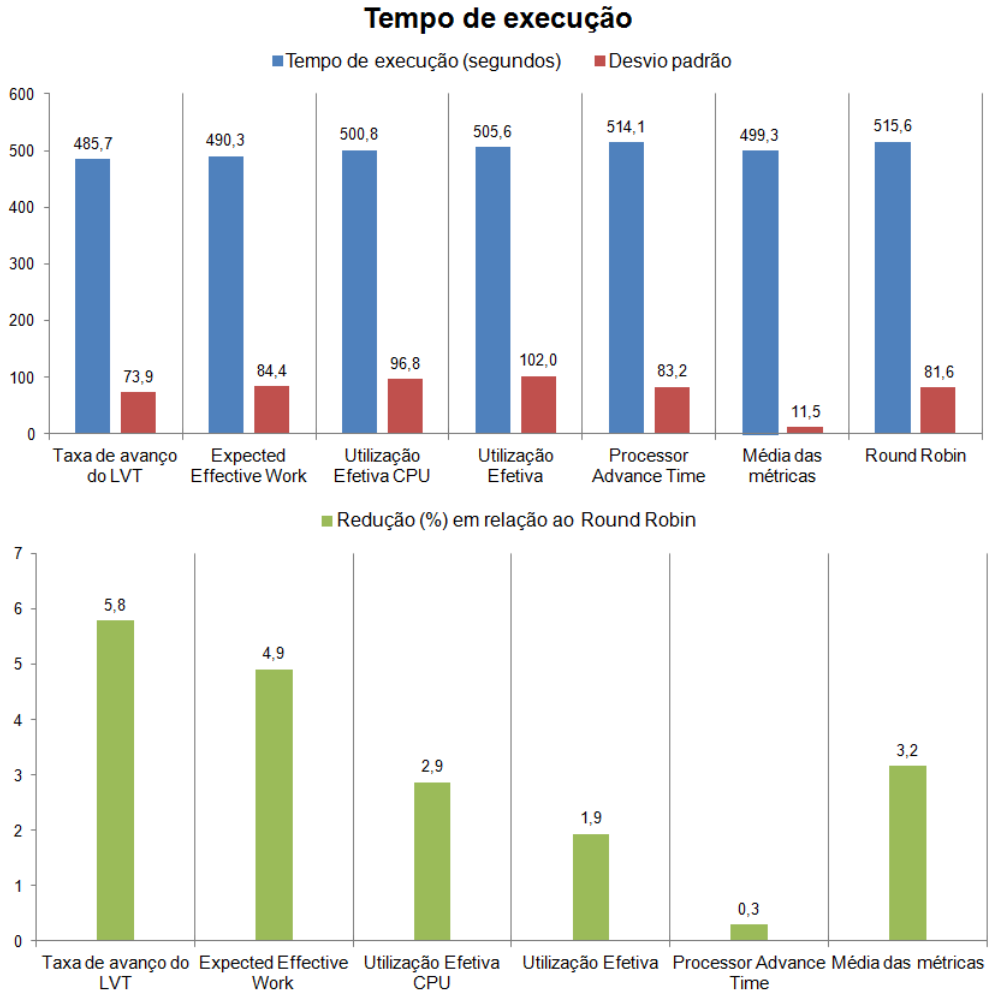

Figura 24: Tempo de execução para cada métrica em relação aos três modelos (grupos, *hnet* e *dnet*)

Durante a execução das simulações, foi notado que a maioria das migrações ocorre no início da simulação. Após um determinado ponto dificilmente houve migrações. Essa característica pode fazer com que a redução no tempo de execução das simulações seja maior a medida que o tempo de execução da simulação cresça, de forma que o tempo gasto para realizar as migrações seja dissolvido nesse

tempo total. Uma alternativa também seria estudar melhorias no mecanismo de migração. Vale lembrar que o mecanismo de migração utilizado nesse trabalho foi o mecanismo de migração coletiva de (JUNQUEIRA, 2012).

Essa análise com todos os modelos é importante para determinar as métricas mais eficientes, independente do modelo. As diferenças notadas nas análises individuais de cada modelo mostraram que cada um foi capaz de produzir diferentes resultados, ou seja, esses modelos puderam alcançar uma abrangência suficiente para uma generalização dos resultados. Essa abrangência pode ser percebida pelo fato de que, para cada modelo, uma métrica diferente alcançou a melhor eficiência.

8 Conclusão

Esse trabalho apresentou uma comparação entre diferentes métricas utilizadas para o balanceamento de carga em ambientes de simulação distribuída otimistas que operam com o protocolo *Time Warp*, bem como a proposta de uma nova métrica de balanceamento. A métrica proposta foi baseada nos principais fatores causadores de desbalanceamento de carga nesses ambientes.

Para realizar a comparação entre as métricas, foi proposto um algoritmo de balanceamento de carga dinâmico que opera independente da métrica utilizada para determinar a carga do sistema. Esse algoritmo foi proposto de forma a não beneficiar ou prejudicar nenhuma das métricas avaliadas.

Ainda com o objetivo de submeter cada uma das métricas a um mesmo ambiente, foram propostos modelos e sementes de aleatoriedade arbitrários, de forma que todas as métricas fossem avaliadas com uma sequência de simulações idênticas. Idênticas no sentido de ser o mesmo modelo, com a mesma conectividade e probabilidades de comunicação entre os processos.

Para cada modelo e métrica diferentes, foram realizadas 10 simulações, cada uma das simulações com os valores dos fatores de desbalanceamento aleatórios. Essa aleatoriedade permitiu que, para um mesmo modelo, cada uma das simulações possuísse uma variação de carga diferente, objetivando afetar a forma de trabalho do mecanismo de balanceamento. A variação da carga imposta ao ambiente de simulação considerou a variação na taxa de geração de eventos, o tempo de serviço e ainda cargas externas à simulação.

Pela análise dos resultados, foi observado que os modelos foram genéricos o suficiente, apresentando resultados diferentes em relação a eficiência, número de migrações e tempo de execução das simulações, para cada um deles.

Com uma análise geral sobre todos os modelos verificou-se que a métrica da taxa de avanço do LVT é a mais eficiente, com uma eficiência aproximadamente 13% superior ao *Round Robin*. As métricas baseadas na utilização efetiva são quase tão eficientes quanto a métrica da taxa de avanço do LVT, com uma eficiência aproximadamente 12,5% superior ao *Round Robin*. Essas três métricas também foram bem semelhantes em relação ao número de migrações que realizaram e aos tempos gastos para execução das simulações. A redução no tempo de execução ficou condicionada ao número de migrações realizadas, pois as métricas que migram mais acabam reduzindo menos o tempo de execução das simulações. Para as métricas que tem uma média de número de migrações iguais, as que possuíam um desvio padrão maior reduziram menos o tempo de execução.

A métrica PAT e a métrica EEW tiveram eficiências abaixo da média de todas as métricas. A métrica EEW, proposta nesse trabalho, teve uma eficiência quase igual a média, apenas 5% abaixo, enquanto a métrica PAT teve uma eficiência 15% abaixo da média. Em relação à métrica mais eficiente, que foi a taxa de avanço do LVT, a métrica EEW obteve uma eficiência 12,5% inferior e a métrica PAT uma eficiência 22% inferior. Vale ressaltar que, apesar de uma eficiência 12,5% inferior, a métrica EEW apresentou uma redução do tempo de execução quase idêntica, menos de 1% inferior.

Os resultados mostram que todas as métricas avaliadas são capazes de tratar todos os fatores de desbalanceamento de carga impostos às simulações executadas, já que todas alcançaram eficiência superior ao *Round Robin*. Mesmo as métricas que não são calculadas a partir de unidades de processamento, ciclos de CPU, são capazes de tratar o desbalanceamento causado por carga externa. Na verdade, tais métricas obtiveram resultados até melhores que as métricas calculadas a partir desse tipo de informação.

Além de ser a mais eficiente, a métrica da taxa de avanço do LVT é a mais simples de ser calculada, pois essa informação já faz parte do ambiente de uma simulação distribuída. Métricas como a PAT e a utilização efetiva com base em ciclos de CPU são mais difíceis de serem calculadas, além de estarem sujeitas a características da plataforma.

Como a avaliação das métricas foi realizada em uma mesma plataforma, com a mesma biblioteca de passagem de mensagens, nesse caso o *OpenMPI*, com os mesmos modelos e com o mesmo algoritmo de balanceamento, ela é considerada genérica o suficiente para a validade dos resultados apresentados.

8.1 Contribuições

A principal contribuição desse trabalho foi a construção de um ambiente de simulação distribuída, com diferentes modelos de simulação, diferentes variações de carga e com um algoritmo de balanceamento capaz de avaliar as métricas de balanceamento sob as mesmas condições, sem beneficiar ou punir qualquer uma delas. Outra contribuição importante foi a avaliação realizada, com modelos e variações de carga que se mostraram genéricos o suficiente para valida-la, mostrando as métricas mais eficientes que podem ser consideradas em projetos de algoritmos de balanceamento de carga par simulações distribuídas.

A métrica EEW e o algoritmo de balanceamento dinâmico, que pode operar com diferentes métricas, também foram uma contribuição significativa. A métrica EEW não foi a mais eficiente, mas ficou próxima à média de eficiência das métricas avaliadas e pode ser otimizada a fim de melhorar seu desempenho.

Uma última contribuição desse trabalho foram as melhorias na implementação do protocolo *Time Warp* e no mecanismo de migração de processos. Essas melhorias foram descritas no Capítulo 4.

8.2 Trabalhos futuros

Novos estudos podem ser desenvolvidos a partir das implementações e análises realizadas nesse trabalho.

Podem ser estudadas algumas possibilidades de melhorias no mecanismo de migração, a fim de melhorar a redução no tempo de execução das simulações.

Uma avaliação de melhoria da métrica EEW ou do algoritmo de balanceamento pode ser realizada, considerando outras abordagens.

Podem ser realizados mais testes, considerando configurações e modelos de simulação diferentes, visando completar ainda mais as análises realizadas neste trabalho.

Pode ser considerada a proposta de um algoritmo de balanceamento capaz de alternar entre as métricas, de forma a usar a métrica que melhor se adapte ao modelo. Como foi visto na análise dos resultados, para cada modelo uma métrica diferente foi a mais eficiente. Um algoritmo capaz de identificar a métrica mais eficiente para cada modelo obteria um resultado superior a um algoritmo que optasse pela melhor métrica avaliada sobre diferentes modelos.

Referências

AGARWAL, A.; HYBINETTE, M.; XIONG, Y. Merging parallel simulation programs. In: *The 18Th Workshop on Parallel and Distributed Simulation - PADS2005*. [S.l.: s.n.], 2005.

AJALTOUNI, E. E.; BOUKERCHE, A.; ZHANG, M. An efficient dynamic load balancing scheme for distributed simulations on a grid infrastructure. In: *Distributed Simulation and Real-Time Applications, 2008. DS-RT 2008. 12th IEEE/ACM International Symposium on*. [S.l.: s.n.], 2008. p. 61–68.

AZEVEDO, M. E. C. V. de. *Implementação e comparação dos protocolos Time Warp e Rollback Solidário*. Dissertação (Mestrado em Ciência e Tecnologia da Computação) — Universidade Federal de Itajubá - Instituto de Engenharia de Sistemas e Tecnologias da Informação, Itajubá - MG, 2012.

BABAOGLU, O.; MARZULLO, K. Consistent global states of distributed systems: Fundamental concepts and mechanisms. *MULLENDER Distributed Systems*, p. 55–96, 1993.

BANKS, J. et al. *Discrete-Event System Simulation*. 4. ed. Índia: Prentice Hall, 2008.

BARAK, A.; SHILOH, A. A distributed load-balancing policy for a multicomputer. *Softw. Pract. Exper.*, v. 15, n. 9, p. 901–913, 1985.

BAUSE, F.; EICKHOFF, M. Truncation point estimation using multiple replication in parallel. In: *The 2003 winter simulation conference*. [S.l.: s.n.], 2003. p. 414–421.

BELAPURKAR, A. et al. *Distributed systems security issues, processes and solutions*. 1. ed. USA: John Wiley & Sons, Inc, 2009.

BRANCO, K. R. L. J. C. *Índices de carga e desempenho em ambientes paralelos/distribuídos - modelagem e métricas*. Tese (Doutorado em Ciências de Computação e Matemática Computacional) — Universidade de São Paulo - Instituto de Ciências Matemáticas e de Computação, São Carlos - SP, 2004.

BRYANT, R. E. Simulation of packet communication architecture computer systems. *MIT-LCS-TR-188 - Massachussets*, v. 7, n. 3, p. 404–425, 1977.

BURDORF, C.; MARTI, J. Load balancing strategies for Time Warp on multi-user workstations. *The Computer Journal*, 1993.

CAROTHERS, C. D.; FUJIMOTO, R. Efficient execution of time warp programs on heterogeneous, NOW platforms. *IEEE Transactions on Parallel and Distributed Systems*, PDS-11, n. 3, p. 299–317, 2000.

Carvalho Junior, O. A. de. *Avaliação de políticas de escalonamento para execução de simulações distribuídas*. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) — Universidade de São Paulo - Instituto de Ciências Matemáticas e de Computação, São Carlos - SP, 2008.

CASAVANT, T. L.; KUHL, J. G. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, v. 14, n. 2, 1988.

CASSANDRAS, C. G.; LAFORTUNE, S. *Introduction to Discrete Event Systems*. 2. ed. USA: Springer, 2008.

CHANDY, K. M.; MISRA, J. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, n. 5, p. 440–452, 1979.

CHEN, L.-l. et al. A well-balanced time warp system on multi-core environments. In: *Proceedings of the 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation*. [S.l.]: IEEE Computer Society, 2011. (PADS '11), p. 1–9.

CHILD, R.; WILSEY, P. Dynamically adjusting core frequencies to accelerate time warp simulations in many-core processors. In: *Principles of Advanced and Distributed Simulation (PADS), 2012 ACM/IEEE/SCS 26th Workshop on*. [S.l.: s.n.], 2012. p. 35 –43.

CHWIF, L.; PAUL, R. J.; BARRETTO, M. R. P. Discrete event simulation model reduction: A causal approach. *Simulation Modelling Practice and Theory EUA*, v. 4, p. 930–944, 2006.

COULOURIS, G. F.; DOLLIMORE, J.; KINDBERG, T. *Distributed systems: concepts and design*. 4. ed. England: Pearson Education, 2005.

De Grande, R. E.; BOUKERCHE, A. Dynamic balancing of communication and computation load for HLA-based simulations on large-scale distributed systems. *Journal of Parallel and Distributed Computing*, v. 71, n. 1, p. 40–52, 2011.

DE GRANDE, R. E.; BOUKERCHE, A. Dynamic balancing of communication and computation load for HLA-based simulations on large-scale distributed systems. *Journal of Parallel and Distributed Computing*, v. 71, n. 1, p. 40–52, 2011.

FUJIMOTO, R. M. *Parallel and distributed simulation systems*. 1. ed. USA: John Wiley & Sons, Inc, 2000.

FUJIMOTO, R. M. Distributed simulation systems. In: *Proceedings of the 2003 Winter Simulation Conference*. [S.l.: s.n.], 2003. p. 124–135.

GLAZER, D. W.; TROPPER, C. On process migration and load balancing in time warp. *IEEE Transactions on Parallel and Distributed Systems*, v. 4, n. 3, p. 318–327, 1993.

JEFFERSON, D. R. Virtual time. *ACM Transactions on Programming Languages and Systems*, v. 7, n. 3, p. 404–425, 1985.

JIANG, M.; ANANE, R.; THEODOROPOULOS, G. Load balancing in distributed simulations on the grid. In: *Systems, Man and Cybernetics, 2004 IEEE International Conference on*. [S.l.: s.n.], 2004. v. 4.

JUNQUEIRA, M. A. F. C. *Mecanismos para Migração de Processos na Simulação Distribuída*. Dissertação (Mestrado em Ciência e Tecnologia da Computação) — Universidade Federal de Itajubá - Instituto de Engenharia de Sistemas e Tecnologias da Informação, Itajubá - MG, 2012.

KASSAB, E. et al. Blowing up the barriers in surgical training: Exploring and validating the concept of distributed simulation. *Annals of surgery*, LWW, v. 254, n. 6, p. 1059–1065, 2011.

KUMAR, V. *Introduction to parallel computing design and analysis of algorithms*. 2. ed. USA: Addison Wesley Publishing Company, 2003.

LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, v. 21, n. 7, p. 558–565, 1978.

LAMPORT, L.; SHOSTACK, R.; PAESE, M. Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, v. 4, n. 3, p. 382–401, 1982.

LAW, A. M. *Simulation modeling and analysis*. 4. ed. USA: McGraw-Hill, 2007.

LINUXMANPAGES. *Clock and time functions*. 2013. `http://linuxmanpages.net/manpages/fedora13/man2/clock_getres.2.html`. Acessado em 27/02/2013.

LINUXMANPAGES. *Getrusage function*. 2013. `http://linuxmanpages.net/manpages/fedora13/man2/getrusage.2.html`. Acessado em 27/02/2013.

LOW, M. Y. H. Dynamic load-balancing for bsp time warp. In: *In Proceedings of the 35th Annual Simulation Symposium*. [S.l.]: IEEE Computer Society, 2002. (SS'02), p. 267–274.

MALIK, A. W.; PARK, A. J.; FUJIMOTO, R. M. Optimistic synchronization of parallel simulations in cloud computing environments. *IEEE International Conference on Cloud Computing*, p. 49–56, 2009.

MATTERN, F. Efficient algorithms for distributed snapshots and global virtual time approximation. *Journal of Parallel and Distributed Computing*, v. 18, n. 4, p. 423–434, 1993.

MEDINA, A. C.; CHWIF, L. *Modelagem e Simulação de Eventos Discretos - Teoria e Aplicações*. 2. ed. [S.l.]: Bravarte, 2007.

MERAJI, S.; ZHANG, W.; TROPPER, C. A multi-state q-learning approach for the dynamic load balancing of time warp. In: *Proceedings of the 2010 IEEE Workshop on Principles of Advanced and Distributed Simulation*. [S.l.]: IEEE Computer Society, 2010. (PADS '10), p. 142–149.

MILOJICIC, D. S. et al. Process migration. *ACM Computing Surveys*, v. 32, n. 3, p. 241–299, 2000.

MOREIRA, E. M. *Rollback Solidário: Um Novo Protocolo Otimista para Simulação Distribuída*. Tese (Doutorado em Ciências de Computação e Matemática Computacional) — Universidade de São Paulo - Instituto de Ciências Matemáticas e de Computação, São Carlos - SP, 2005.

PESCHLOW, P.; HONECKER, T.; MARTINI, P. A flexible dynamic partitioning algorithm for optimistic distributed simulation. In: *Principles of Advanced and Distributed Simulation, 2007. PADS '07. 21st International Workshop on*. [S.l.: s.n.], 2007. p. 219–228.

PITANGA, M. *Computação em cluster*. 1. ed. São Paulo: BRASPORT, 2003.

RAVASI, J. F. *JUMP: Uma política de escalonamento unificada com migração de processos*. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) — Universidade de São Paulo - Instituto de Ciências Matemáticas e de Computação, São Carlos - SP, 2009.

REIHER, P. L.; JEFFERSON, D. Virtual time based dynamic load management in the time warp operating system. *Transactions of the Society for Computer Simulation*, v. 7, n. 2, p. 103–111, 1990.

SCHRIBER, T. J.; BRUNNER, T. D. Inside discrete-event simulation software: How it works and why it matters. In: *Proceedings of the 34th Winter Simulation Conference*. [S.l.: s.n.], 2002. v. 1, p. 97–107.

SILBERSCHTZ, A.; GALVIN, P. B.; GAGNE, G. *Operating System Concepts*. 8. ed. USA: John Wiley & Sons, Inc, 2009.

SOM, T. K.; SARGENT, R. G. A probabilistic event scheduling policy for optimistic parallel discrete event simulation. In: *Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS-98)*. [S.l.]: IEEE Computer Society, 1998. p. 56–63.

SOM, T. K.; SARGENT, R. G. Model structure and load balancing in optimistic parallel discrete event simulation. In: *Procedings of the 14th Workshop on Parallel and Distributed Simulation (PADS-00)*. [S.l.]: IEEE Press, 2000. p. 147–156.

SOUZA, P. S. L. *AMIGO: Uma contribuição para a convergência na área de escalonamento de processos*. Tese (Doutorado em Física Aplicada Opção Em Física Computacional) — Universidade de São Paulo - Instituto de Física de São Carlos, São Carlos - SP, 2000.

TANENBAUM, A. S. *Sistemas Operacionais Modernos*. 3. ed. São Paulo - SP: Pearson Prentice Hall, 2010.

TANENBAUM, A. S.; STEEN, M. V. *Distributed systems: principles and paradigms*. 2. ed. Upper Saddle River: Prentice Hall, 2006.

VOORSLUYS, B. L. *Influências de políticas de escalonamento no desempenho de simulações distribuídas*. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) — Universidade de São Paulo - Instituto de Ciências Matemáticas e de Computação, São Carlos - SP, 2006.